IEEE MICRO

Chips, Systems, Software, and Applications

APRIL 1991

OPTICAL ARCHITECTURE

Efficient interprocessor communication

Cache memory design

Simulating DSP-based petri nets

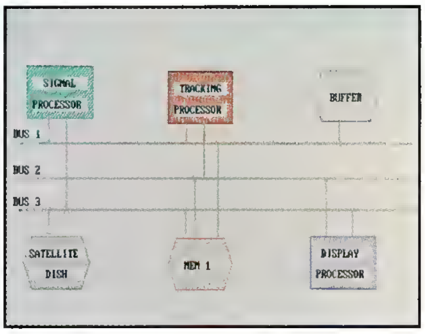
F SERVICE
ER SOCIETY

ELECTRICAL AND
INEERS, INC.

Six Simulation Solutions

Now you can predict system performance **before** costly implementation

NETWORK II.5

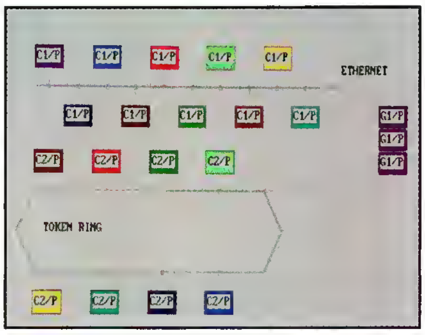


Predict computer/communications network performance. You graphically define your network of processors and storage devices, interconnected by communication channels. Animation follows immediately—no programming.

COMNET II.5

Predict LAN, LAN/WAN, voice, and data network performance. You graphically define the topology of your network through routing nodes and transmission links. Animation follows immediately—no programming.

LANNET II.5



Predict LAN performance. You graphically define your network of stations and gateways interconnected by LAN's. Animation follows immediately—no programming.

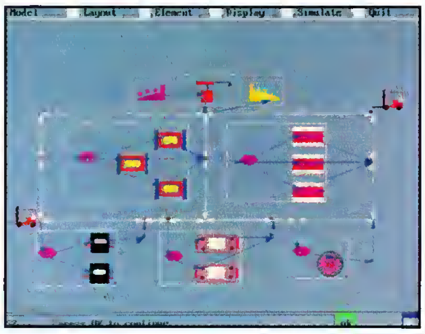
SIMSCRIPT II.5

A widely used programming language for development of process-oriented models. This English-like language, with built-in graphics, is in use at 4,636 sites worldwide.

MODSIM II

A new object-oriented programming language with built-in graphics. It incorporates the key elements of modern software engineering: block structure, modularity, and strong typing.

SIMFACTORY II.5



Predict factory performance. You graphically describe your factory through an easy-to-use interface. Animation follows immediately—no programming.

Free Trial and Training Offer

For over 29 years CACI has provided its simulation software on a free trial basis—**no cost, no obligation**. You get everything you need to try the product of your choice on your PC, Workstation, or Mainframe.

For Immediate Information:
In the U.S., call (619) 457-9681 or Fax (619) 457-1184. In the UK or Europe, call Nigel McNamara, in our UK office, on (081) 332-0122 or Fax (081) 332-0112. In Canada, call Peter Holt, at (613) 782-2474, Fax (613) 782-2202.

Rush free trial and training information on:

☐ SIMSCRIPT II.5® ☐ MODSIM II®
☐ SIMFACTORY II.5® ☐ NETWORK II.5®
☐ COMNET II.5® ☐ LANNET II.5™

☐ Send me information on your special university program for faculty only.

Name ____________________
Company ____________________
Address ____________________
____
____
Telephone __________ Fax __________
Computer __________ Op. Syst. __________

Mail or Fax this coupon to:
CACI Products Company
3344 North Torrey Pines Court
La Jolla, California 92037
Phone (619) 457-9681, Fax (619) 457-1184

In the UK or Europe:
CACI Products Division
Palm Court, 4 Heron Square
Richmond, Surrey, TW9 1EW
United Kingdom
Phone (081) 332-0122, Fax (081) 332-0112

In Canada:
CACI Products Division
200-440 Laurier Avenue West
Ottawa, Ontario K1R 7X6
Phone (613) 782-2474, Fax (613) 782-2202

IE MICRO

IEEE MICRO

1991 Editorial Calendar

FEBRUARY

Microprocessors in Education
Special 10th Anniversary issue

- Demonstrating new hardware to students
- Part 3, legal issues surrounding the *Lotus* vs. *Paperback* decision
- Industry prospects in the next 10 years

Ad closing date: January 1, 1991

AUGUST

Special Far East issue

- Microcomputer and RAM technology
- CISC and superscalar
- High-performance parallel computer research
- Current TRON Project offerings, Japan's standard computer system

Ad closing date: July 1, 1991

APRIL

General interest

- Optical multiprocessor architecture
- Efficient interprocessor communication
- Cache memory design
- Simulating DSP-based petri nets
- Report from Japan: The Sixth Generation Project

Ad closing date: March 1, 1991

OCTOBER

Computers in bioengineering

- Medical imaging
- Robotics in surgery
- Monitoring and measuring biological processes
- Special feature: Fuzzy logic projects in Japan

Ad closing date: September 1, 1991

JUNE

Hot Chips II Symposium

- This extremely popular issue presents the latest developments in microprocessor and chip technology used to construct high-performance workstations and systems.
- Contributors include: Metaflow Technologies, IBM Research, Intergraph, ITT Intermetall, and Intel

Ad closing date: May 1, 1991

DECEMBER

Database machine implementation

- Ensuring that today's microcomputers efficiently implement databases
- Architecture design
- More to be added

Ad closing date: November 1, 1991

IEEE Micro helps designers and users of microprocessor and microcomputer systems explore the latest technologies to achieve business and research objectives.
Feature articles in *IEEE Micro* reflect original works relating to the design, performance, or application of microprocessors and microcomputers.
All manuscripts are subject to a peer-review process consistent with professional-level technical publications. *IEEE Micro* is a bimonthly publication of the IEEE Computer Society.

Advertising information: Contact Marian Tibayan, Advertising Department, IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; (714) 821-8380; fax (714) 821-4010.

Articles may change. Please contact editor to confirm.

Letters

Send letters to
Managing Editor
IEEE Micro
PO Box 3014
Los Alamitos, CA
90720-1264

Precisely speaking ...

To the editor:

Thank you for another DSP special issue (Oct. 1990). I am definitely convinced about the importance of this particular topic. Unfortunately, it seems that a little inaccuracy appeared in "Programmable DSPs: A Brief Overview" by Edward A. Lee.

Several bulletins and other technical literature available from TI claim the 320C25 to be an enhanced CMOS version of the 32020, with internal ROM, increased execution speed, extended instruction set, and other facilities. The 320C26 in turn differs from the 320C25 only in internal memory organization and on-chip RAM configuration instructions. To be precise, TMS320C25 contains 4K ROM + 544 RAM words, and TMS 320C26 contains 256 ROM + 1.5K RAM words.

Jerzy R. Chrzaszcz
Warsaw

The author replies:

This observation is correct. Thanks for pointing it out.

Edward A. Lee
Berkeley, CA

Improving the product

To the editor:

Comments on the December 1990 issue: I find the splitting of printing the papers in the issue highly inconvenient and frustrating. Please return to continuous printing of articles as you used to.

Comments on IBM RS/6000 benchmark data, bottom of page 52: The data are not that significant since comparison with the really top runners was not made. It would be more significant if the RS/6000 were compared with microprocessors that are in its own league by performance: among the CISC microprocessors, the MC68040, and among the RISC-type, the Intel 80860. A comparison with the Intel superscalar 80960 would also be interesting.

Daniel Tabak
Fairfax, VA

The editors reply:

We elected to split articles for two reasons, cost cutting and reader convenience. Placing all color pages in an issue in one section helps us cut printing costs. Splitting Micro*'s generally long articles after the first four pages gives readers the option of sampling each article and returning later to those especially appealing to them for a full reading.—M.E.*

Your comments on the Micro News benchmarking item present quite an interesting approach. Perhaps the research laboratory that supplied the data will act on your suggestions. If it does, we will be happy to print it. I appreciate this (and similar) comments that may help readers to better understand the information displayed in IEEE Micro.*—D.D.C.*

Expedited delivery

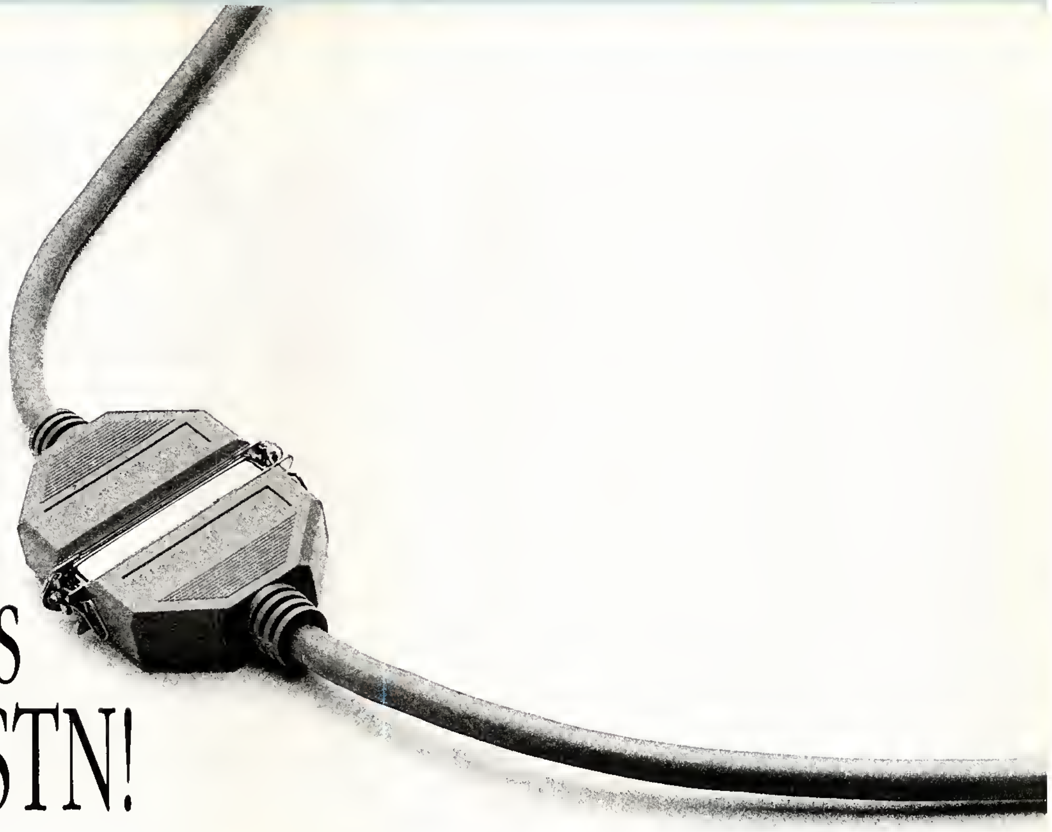
is available to all members residing outside the USA, Canada, and Mexico. We invite you to take advantage of this service providing delivery of your magazines weeks earlier.

♦

For information on this service and its cost, contact:

Expedited Delivery
IEEE Computer Society
10662 Los Vaqueros Circle
PO Box 3014
Los Alamitos, CA 90720-1264 USA
(714) 821-8380
FAX (714) 821-4010

Make Great Connections Online On STN!



On STN International,® you can access databases covering every area of engineering. You'll find bibliographic and numeric files produced by leading scientific organizations, like AIChE, Engineering Information, IEEE, National Institute of Standards and Technology, and many others.

And everything about STN has been created to assist you in obtaining this information efficiently and economically. On STN, you'll find special features which enhance your searching, whether you're a novice or an expert.

One Command Language — Using a few simple commands, you'll be able to obtain information in more than 100 databases on STN. And STN's software, Messenger,® enables you to carry a search created in one database over to another on STN for further information.

Element Term Index — Through STN's Element Term (ET) Field, you can search for chemical symbols and other specialized notations. Using the ET Field can increase your accuracy AND efficiency.

Numeric Searching — On STN, you can search numeric values to locate substances having the property values you specify; search numeric ranges to find data you might otherwise miss; choose from SI, metric, engineering, or other units to display property values; search with substance names, names of properties, or CAS Registry Numbers® to retrieve numeric data.

As an STN customer, you can receive help from workshops, tutorial diskettes, STN Express® software, toll-free Help Desk, newsletters, and online document ordering. No one supports you like STN!

Make great connections on STN by filling out and returning the coupon below.

☐ **YES! Please tell me how to become a user of STN International.**

Name

Title

Organization

Address

City, State ZIP

Phone

MAIL TO: STN International, c/o Chemical Abstracts Service, Marketing Dept. 30091, P.O. Box 3012, Columbus, OH 43210
FAX TO: 1-614-447-3713

STN® INTERNATIONAL
The Scientific & Technical Information Network

IEEE MICRO

Published by the IEEE Computer Society

April 1991

SPECIAL FEATURES

Coming Next Issue: Hot Chips II

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; IEEE Computer Society Headquarters, 1730 Massachusetts Ave., NW, Washington, DC 20036-1903; IEEE Headquarters, 345 East 47th St., New York, NY 10017. Annual subscription: $21 in addition to IEEE Computer Society or any other IEEE society member dues; $38 for members of other technical organizations. This journal is also available in microfiche form.
Postmaster: Send address changes and undelivered copies to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. Second-class postage is paid at New York, NY, and at additional mailing offices.

DEPARTMENTS

IEEE Computer Society
PO Box 3014, Los Alamitos, CA 90720-1264
(714) 821–8380

* Submit six copies of all articles and special-issue proposals to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129 Torino, Italy; phone +39-11-556-7244; Internet: delcorso@pol88B.to.cnr.it; Bitnet: delcorso@itopoli;
or
** Ashis Khan, Mips Computer Systems, Inc., 950 DeGuigne Drive, Sunnyvale, CA 94086; Internet: ashis@mips.com.

From the Editor-in-Chief

The ultimate solution?

THE MOST IMPORTANT news for this issue is that *IEEE Micro* has a new associate editor-in-chief: Ashis Khan accepted a proposal that he share this honor and (most importantly!) the associated work. Readers in the US can easily reach him at Mips Computer Systems, Inc., 950 DeGuigne Drive, Sunnyvale, CA 94086; telephone (408) 524-7171, fax (408) 524-7952, and e-mail: ashis@mips.com. For information on submitting papers, general comments, proposals, and even complaints, you may interact with either of us, whoever you prefer.

Next are some of my thoughts on this issue. We originally planned it as a general-interest issue, without a specific theme. However, as I reviewed the individual articles—which deal with important aspects of microsystems—I began to see a common frame emerge. I realized that these articles address a rather "hot" theme: communication in distributed processing systems. As integration technology continues to let us place more transistors in fewer square millimeters, it becomes more evident that the performance bottleneck occurs not in the elementary processing elements but in the subsystem that controls the exchange of information among them.

Communication procedures and technology did not evolve fast enough to cope with the power and complexity of processors. The simplest way to realize this fact is to think about the amount of gates that increases proportionally to the square of the chip/feature size ratio (chip area/transistor size), while the communication capability (the number of I/O pads) increases only linearly. Designers must compensate for this imbalance with more efficient and sophisticated procedures for information exchange.

The authors in this issue accepted this challenge, and we benefit from their experiences. The first article, "System Effects of Interprocessor Communication Latency in Multicomputers" by Xiaodong Zhang, analyzes the effects of architectures, topology, and communication methods. The next article discusses caching, one of the techniques used to lighten the load on the communication system of a multiprocessor. In "RST Cache Memory Design for a Tightly Coupled Multiprocessor System" Cosimo Prete presents a cache coherence protocol with interesting new features.

Another approach reflects that the selection of a specific communication mechanism requires proper design tools. Petri nets, used by many researchers to model memory hierarchies, are also used generally to evaluate the efficiency of different multiprocessor architectures. "Implementing a DSP-Based Petri-Net Simulation Tool" by Antonella Di Stefano, Orazio Mirabella, and Fabio Presente discusses this approach. The authors present a hardware accelerator that permits high-speed simulation of loosely and tightly coupled architectures.

However, the ultimate solution for the communication bottleneck lies in moving from charged particles (electrons) to photons, that is,

0272-1732/91/0400-0006$01.00 © 1991 IEEE

switching to optical computing. This is the theme of the cover and of the last article, "3D Optical Architecture and Data-Parallel Algorithms for Massively Parallel Computing." Here, Ahmed Louri discusses a direction of future development in parallel computing. (This theme is worth a special issue by itself; does anyone care to present a proposal for it?)

[illegible]

In the mailbag

What you see reproduced here constitutes truly international feedback! All comments were received by mid-February.

(LK: liked, DLK: disliked, LTS: like to see)

April 1990

LK: MMU review, the TRON intelligent house, Micro View; LTS: more about fault tolerance—R.S., Tehran, Iran

LK: The TRON intelligent house—D.F., Argentina

LK: The TRON intelligent house; LTS: more points of view on same topic—D.S., Kewdale, Australia (TRON received a high score; we will publish other issues on this theme.—D.D.C.)

LK: All features; LTS: more DSP.—T.A., Istanbul (We plan another DSP issue next year.—D.D.C.)

LK: Parallel processor element design; LTS: more practical design approach—M.S.P., Pune, India

June 1990

LK: Gmicro/300 ..., 88000 family, Micro Review; LTS: recommendation of reference books on RISC.—M.R.R, Belgrade, Yugoslavia (Look at the references at the end of the articles in the June 1991 Hot Chips issue—D.D.C.)

LK: Micro Review—A.S.M., Secunderabad, India

LK: Best issue since the start of my subscription. The subject is hot ... 68040 [article] is excellent ... my very special thanks to the authors; LTS: more articles written in this style—G.M., Warsaw

LK: Micro View, we should always refresh our knowledge. DLK: Jumping across articles; LTS: video processors—A.V.C., Valparaiso, Chile (I strongly agree with your comment on knowledge. Regarding the article jumps, see the answer to Tabak in this issue's Letters column.—D.D.C.)

August 1990

LK: Articles on patent law; DLK: Generally this was a boring issue; LTS: More articles on pipelined microprocessors and architectures, superscalar, dynamic instruction scheduling—R.S., Tucson, AZ (Some disagreement, at last; readers, please send in your negative comments as well as the positive ones! R.S., your proposals and suggestions for articles are fully acknowledged in the June 1991 Hot Chips issue.—D.D.C.)

DLK: The practically unreadable titles in color ... contents page on back of front cover—C.J.D., Whittier, CA (The color problem has been corrected. We often print the contents on this page to keep costs down.—D.D.C.)

LTS: More OS/2 and presentation manager—T.B., Houston, TX

LK: New Products, Micro News, Micro Review; DLK: The whole issue was not very interesting to me. LTS: Advanced programmable logic—T.P., Warsaw

LTS: How can I get a copy of IEEE standards?—A.V.C., Valparaiso, Chile (Anyone interested in buying copies of IEEE standards should contact the IEEE Service Center at PO Box 1331, Piscataway, NJ 08855-1331, telephone (908) 981-0060, or fax (908) 981-0027.—D.D.C.)

October 1990

LK: DSP-related articles; LTS: more on environments for engineering design—R.N.Z., Manchester, UK

LK: Original aspects of current chips; LTS: more about new products—I.K., Yugoslavia (Watch for the upcoming Hot Chips issue.—D.D.C.)

LK: Presentation and articles; LTS: info packet, USAT, modem X25—E.P.A., Chile

LK: From the Editor-in-Chief ... very good issue—A.V.C., Valparaiso, Chile (This comment is credit for Stephen Dyer and Richard Higgins, guest editors of this issue.—D.D.C.)

LK: Overview article ... very informative; LTS: more articles on architecture design on PCs.—S.K., Kajang, Malaysia

LK: Micro View ... computer architectures; LTS: IC announcements—it is more logical to place them in *Micro* than in *Computer* magazine.—G.M., Warsaw (*Computer* is the magazine every member of the IEEE Computer Society receives; everybody who receives *Micro* receives *Computer*.—D.D.C.)

LK: Guest editor's introduction and brief overview, but in general not a howling success. The previous DSP special issue was better. LTS: More on DSP hardware—J.R.C., Warsaw

Micro News

Send information for inclusion in Micro News one month before cover date to Managing Editor, IEEE Micro, PO Box 3014, Los Alamitos, CA 90720-1264.

HDTV faces intertwined challenges

Ware Myers, Contributing Editor

After years of talk about high-definition television, sets went on sale in Japan last year for $34,000. For that price a consumer receives one hour a day of 1,125-line programming broadcast by NHK (the Japan Broadcasting Corporation) from a satellite.

The Japanese viewer is also treated to a 16:9 aspect ratio—similar to a movie screen's proportions—instead of the present 4:3. And, because the resolution is higher, the screen can be much larger.

Later this year when NHK lofts another satellite, it plans to expand programming to eight hours a day. Then viewer demand will probably pick up. With greater production, prices can drop, perhaps to about $7,500 within five years.

The inauguration of high-definition broadcasting may be one factor behind the acquisition of several Hollywood movie and television production companies by Japanese manufacturers. They need high-resolution programming if they are to sell high-definition television sets.

Despite this important milestone, the path to HDTV in every home appears to twist through a variety of interrelated challenges. The first one is the viewer.

Viewer desires. Do viewers want HDTV enough to pay higher prices for it? Will broadcasters and other entities invest in the means needed to transmit it? To put the issue in reverse: How far down can technology, standards, and mass production drive the initial high cost? The Japanese are about to get at least preliminary answers to these questions.

Television receivers currently operate at the US NTSC standard of 525 lines or European PAL and SECAM standards of 625 lines. (NTSC refers to the US National Television System Committee standard used by 32 countries; PAL is the phase alternate line standard for 63 countries; SECAM is the sequential color and memory standard for 42 countries.)

At these levels the picture does not have the same satisfying sharpness or solidity that we see in real life. Still, American viewers when visiting Europe often remark how much better the 625-line picture appears, and HDTV would approximately quadruple these resolutions. The number of lines recommended in various proposals ranges from 1,050 to 1,250 with a comparable increase in the number of pixels along a line.

Transmission limits. Terrestrial broadcasting now takes 6 MHz of radio frequency to broadcast low-definition picture and color information and sound. In addition, to minimize interference, an unused channel is spaced between each two channels that are used. HDTV broadcasting, uncompressed, would take about 30 MHz. In the US, the Federal Communications Commission has ruled that broadcasters must serve the 160 million NTSC sets, continuing to occupy considerable broadcast bandwidth. The FCC has also indicated that HDTV must be squeezed within the current terrestrial broadcast bands, VHF and UHF.

The conceptual solution to this complication is data compression, plus more efficient use of the existing bands. Of course, it would take more electronics, both to compress and to reconstitute the data stream. Moreover, much of the electronics would have to be digital, because data compression occurs in that realm.

Three of the teams getting ready to submit systems for testing by the FCC have announced digital systems development. These teams are Zenith Electronics Corporation and American Telephone & Telegraph Co.; Thomson SA of France, Philips NV of The Netherlands, the David Sarnoff Research Center in New Jersey, and NBC;

0272-1732/91/0400-0008$01.00 © 1991 IEEE

and General Instrument Corporation and Massachusetts Institute of Technology. The fourth contestant is the Japan Broadcasting Corporation.

The general idea is to use compression to reduce the quantity of data to be transmitted and, by broadcasting in digital form, to provide a signal less susceptible to interference, permitting the now unused channels to be utilized. The FCC plans to select one of the schemes by 1993.

In addition to terrestrial broadcasting, we use four other transmission mechanisms: direct broadcast satellites, coaxial cable, fiber optics, and videocassettes. Each of them presents its own complications.

Direct-broadcast satellites could be used to transmit HDTV to home receivers, as the Japanese and the British are doing. They expect to continue low-definition service with their existing terrestrial systems. Satellite broadcasting is assigned to much higher frequencies than terrestrial broadcasting. More spectrum space is available there. To receive direct-broadcast signals, however, a user needs a special receiving dish and a decoder.

Coaxial-cable networks are another source of additional bandwidth. In the US, 65 networks offer cable service to about 80 percent of all homes, though only about 60 percent subscribe. Systems now in use have bandwidths of 300 to 400 MHz, providing up to 80 low-definition channels. Bandwidths up to 550 MHz also exist.

Still more capacity could be provided by substituting fiber optics for coaxial cable. Lines already in commercial use can transmit 565 Mbits/second and rates as high as 16 Gbits/second are being developed.

In addition, telephone systems are converting to fiber optics. In the US most of the long-distance lines are already in fiber, but the local lines remain twisted-pair copper wire with only enough bandwidth to carry voice. A full fiber optics network would have enough bandwidth to carry telephone, radio, computer communications, HDTV—everything.

Converting to fiber optics will take enormous investments—about $250 billion in the US and $200 billion in Japan. Making this conversion may take 15 to 40 years, depending on how fast income rolls in to support further investment. Probably only the telephone companies have resources on that scale. Cable companies might then lease bandwidth.

High-definition videocassettes bypass the channel-bandwidth problem. But that leads to a conundrum: No business exists until there is a source of high-definition supply and a market of high-definition receivers. By themselves, cassette manufacturers and retailers can hardly create this industry.

Out there in consumerland, everybody already has a 525-line or 625-line receiver. People want to continue to use it until they are ready to upgrade to an HDTV receiver—a process that will take about 15 years, judging from experience with the switch from black-and-white to color sets. The conundrum again: In the early years who is going to pay for producing and distributing high-definition programming to this small audience?

Design needs. Since not everything is known about these complications, additional research is needed. An uncompressed HDTV signal in digital form will comprise about 1.2 Gbits/second. To process this more than a billion bits may take two or three times that many mathematical operations per second. Processing them will require specialized architectures, DSPs, and microprocessors. Holding them while they are displayed may require 32 Mbytes of memory.

Mass storage of this immense dataflow on videotape or optical media runs into Gbits/second numbers. Transmitting this flow, even on a high-bandwidth medium such as fiber optics, will require new developments in switching and control mechanisms.

To take advantage of the high resolution of these systems, viewers are likely to prefer much larger displays, perhaps 40 to 72 inches diagonally. CRTs in these dimensions are heavy, bulky, yet fragile. The ratio of depth to display area is high. The CRT uses high voltages, emits radiation, and consumes considerable energy. These disadvantages lead to flat-panel displays, such as liquid-crystal or electroluminescent devices, but considerable development work lies ahead before displays of this size become practical.

The road to widespread use of HDTV appears to be a long one bordered by extensive research, design, development, and investment.

Fellowships available

Selected graduate students "whose research has the general goal of improving US industrial competitiveness" will be supported by the recently established Robert Noyce Memorial Fellowships. Two candidates each at four designated schools will receive the annual awards.

The Intel Corporation Foundation, other corporations and foundations, and private individuals combined resources to fund the fellowships in honor of Noyce, the cofounder of Intel and Fairchild Semiconductor Corp. and chief executive officer of Sematech. Applicants should contact the Foundation, 5200 NE Elam Young, Hillsboro, OR 97124 for details.

Measuring MMICs

"We do not have a clear understanding of the measurements we are making, and manufacturers use a variety of different calibration techniques." This statement from Dylan Williams, the US National Institute of Standards and Technology engineer, zeroes in on the problems facing monolithic microwave IC designers.

While the impact of MMIC chips promises to be comparable to that of conventional ICs, present-day designs still must resolve key measurement issues before MMICs can become a commercial technology. "Making conventional ICs is fairly well understood, but at microwave frequencies very different problems arise," says Dennis Friday, assistant chief of NIST's Electromagnetic Fields Division. "Just connecting two MMIC devices together without losing or degrading the signal through reflections is difficult," Friday continued.

One problem is temperature; if MMICs get too hot, they may fail. Yet no proven methods for measuring the temperature of many small MMICs exist. Evaluation methods being examined include infrared micro-radiometry, liquid crystals, and computer modeling. Another problem concerns product uniformity and interconnectability. These essential issues must be solved if manufacturers want to predict accurately how two chips will operate when connected together or to an antenna.

NIST, the US Defense Advanced Research Projects Agency, and an industry-government consortium are working toward making improved measurement and manufacturing technology available as quickly as possible. Friday feels, "The potential here is not just for reducing the size and lowering the cost of microwave devices. MMICs should open up a whole new communications spectrum ... from 50 GHz right up to infrared frequencies." That technology could produce products we can't even guess at now.

continued on p. 83

Reader Service Number 1

Software Report

Sixth Generation Project

David K. Kahaner

US Office of Naval Research, Far East

kahaner@cs.titech.ac.jp

A two-day workshop held December 1-2 in Hakone, Japan, let attendees discuss aspects of the possible 10-year program to follow the Fifth Generation (ICOT) program that ends in 1992. This most-exciting, and largest, program is called New Information Processing Technology. (Westerners call it the Sixth Generation Project.)

Briefly, NIPT would support research and development of new paradigm information processing technologies based on "soft (flexible) information processing" and "integrated computing." The actual meanings of these terms are vague enough that a great deal can be subsumed under them. If this program goes forward as its proponents hope, it will be funded over 10 years at US$30-40 million per year beginning in 1992.

At this time it is not possible to know what form NIPT will take, or even if it will definitely be funded. On the other hand, a final report due March 1991 may define the program well enough that it can be funded coherently. A feasibility study will continue through most of 1991.

At the moment NIPT seems a long way from becoming a coordinated project and looks more like a general umbrella under which a large number of research topics will be covered. The optical computing portion will certainly go forward. The integrated computing section, in which most of the software research is centered, will probably be supported, though I feel it needs to be more clearly defined first. If a third portion on massive parallelism results in an effort to design and build a very large system, that activity will attract world-class researchers like bears to honey. Aspects of the NIPT program probably overlap significantly with the US High-Performance Computing Initiative.

Most of the foreign attendees commented that major aspects of this program are still very vaguely defined. In fact, with the exception of some industrial research projects, the majority of the factual information that was contributed seemed to come from outside Japan. Partially this was because the researchers came prepared to talk about specific research activities, and the Japanese seemed to be more interested in discussing the general directions of the program.

Integrated computing. The workshop combined general sessions with parallel sessions, one of which concerned integrated computing. This track—for systems with advanced architectures using neural and optical computing technologies—was very poorly focused. It was not clear if any concrete ideas have crystallized yet, particularly in the software area. However, Okabe from the University of Tokyo articulated several perfectly reasonable directions that research into neural networks might take in the next 10 years.

Okabe suggested three specific research areas:

- inclusion of the structural development process into conventional algorithms,
- evolutional algorithms such as genetic or chaotic ones, and
- self-organization of structured neural networks.

Okabe also gave one view of the system image of a neural network front and back end to a massively parallel processor (MPP), which would be rule based and focused on symbolic processing. The MPP might be a heterogeneous combination of neural structures, including layered circuits that may be randomly interconnected, completely connected, completely connected in a treelike manner, and dynamically connected.

continued on p. 84

System Effects of Interprocessor Communication Latency in Multicomputers

An important factor in the efficiency of a distributed-memory multicomputer is the effectiveness with which data can be exchanged among its many nodes. A series of experiments and analyses on five types of hypercube and grid-topology multicomputers helped to evaluate interprocessor communication performance. Examination and comparison of system communication speed, message routing, interprocessor connectivity, and software/hardware protocols for passing messages among the five multicomputers enhanced the analysis.

Xiaodong Zhang

University of Texas at San Antonio

Parallel processing applies a simple idea: A computing job can be divided into several tasks that may be executed in parallel. Over the last 10 years designers implemented this concept using distributed-memory multicomputers in a variety of forms in different applications. This experience shows that parallel processing does not reach its anticipated speed when a large number of processors are used in solving problems.[1,2] The communications of common-state information among processors cause a major degradation of the performance (speed).

The literature[3-5] records efforts to measure and evaluate the interprocessor communication performance on the Intel hypercube and the Ncube multicomputers. In addition, Saad and Schultz[6] present several efficient algorithms for data communication on a hypercube multicomputer.

This article takes a wider view, studying various system effects of interprocessor performance, including communication speed, message routing, interprocessor connectivity, and message-passing software/hardware protocols. Both analytical and experimental results offer a clear and comprehensive understanding of the various effects, which is important for the effective use of a distributed-memory multicomputer.

Five multicomputer architectures

In a distributed-memory multiprocessor system, or multicomputer,[7] each processor has its own local memory, and tasks on separate processors coordinate their activities by sending messages through an interconnection network. However, many recent commercial distributed-memory systems vary in computing power, number of processors, type of processors, and network interconnection topology, as well as communication hardware and software.

The hypercube is one example of a distributed-memory, message-passing multicomputer. In a hypercube network 2^n processors are consecutively numbered 0 through $2^n - 1$. Each processor connects to all of the other processors, whose binary representation differs from its own by exactly one bit. This arrangement results in a network that is connected densely enough to support efficient communication between arbitrary processors. Another virtue of the hypercube network is its flexibility: Many other interconnection topologies, such as rings and trees, can be embedded in the hypercube. The dimension n of a hypercube with 2^n nodes determines the maximum number of hops needed to send messages between two nodes. Some system parameters of the five studied multicomputers are:

0272-1732/91/0400-0012$01.00 © 1991 IEEE

- **Intel iPSC/1.** The iPSC/1, one of the first commercially available hypercube computers, may support up to 128 nodes. Each node includes an 8-MHz Intel 80286 processor and 512 Kbytes of local memory. The node operating system supports message-routing asynchronous communications and multitasking within each node.[8]
- **Intel iPSC/2.** This second-generation hypercube features a 4-million-instructions-per-second Intel 80386 node processor, which is four times faster than the 286. Each node can access up to 16 Mbytes of local memory, whereas the iPSC/1 accesses 0.5 Mbytes. The NX/2 operating system supports the new message-passing protocols in the iPSC/2 besides providing a normal system environment in each node.[9]
- **Ncube/10.** This first-generation hypercube system supports up to 1,024 processors. The 32-bit, custom-chip node processor operates at a 7-MHz clock rate and contains 128 Kbytes of local memory. Since the processor includes communication channels, the number of chips per node on the Ncube is relatively low. The Axis operating system supports the transmission of messages between arbitrary nodes of the Ncube/10.[3,10]
- **Ametek 2010.** The Ametek 2010 multicomputer system is based on a 2D grid topology. Each node includes a 25-MHz Motorola 68020 processor and up to 8 Mbytes of local memory.
- **Topology 1000.** This parallel system is a transputer-based variable topology board. The interprocessor network of this Topologix system can be reconfigured. The processor in each node of the network uses the 32-bit, 20-MHz Inmos T800 transputer and up to 16 Mbytes of local memory per processor node.[11-13] The transputer's links are based upon point-to-point interprocessor communication, which eliminates bus contention when messages are transferred. Logix OS is the distributed Unix-compatible operating system supported on the Topology 1000. The Trollius operating system developed at the Cornell Theory Center forms the basis of the Logix OS.

Table 1 summarizes the five types of architectures.

Interprocessor communication

Communication efficiency, one of the most important factors to be considered when designing a multicomputer architecture, often becomes one of the main obstacles to increased performance of parallel algorithms on distributed systems. When a message passes between a pair of nodes in a network, it may be routed through a connected circuit in a number of hops. In addition, intermediate processors may be interrupted to store and then forward the message, or the message may be directly transferred by communication-processing data links through a connected circuit. Thus, the communication speed of the interprocessor network depends on the communication-routing protocols, processor speed, data link speed, and topology of the network.

A comparison of the various effects of different routing models, different interprocessor connections, and other factors to the performance of interprocessor communication on the five types of distributed memory architectures follows.

Communication models. Consider the store-and-forward mechanism[14] used as a typical communication model for first-generation multicomputers such as the iPSC/1, Ncube/10, and Ametek/14. In this communication model, messages pass indirectly between a pair of nodes that are not directly connected via other connected nodes. Each node in the communication path temporarily stores the message in its memory. The processor on each node in that path interrupts work on a task to forward the stored message either to its neighbor or the destination node. Thus, while messages move between a pair of nodes across the network, memory bandwidth and computing cycles in the intermediate nodes are consumed.

The communication latency of this model is also very sensitive to the distance a message must be passed, or it is linearly proportional to the number of hops of the communication. We can express the communication latency of the store-and-forward model as:

$$T_{lat1} = T_{d1} H \qquad (1)$$

where $T_{d1} = K/B_1$, which is the time for a message of size K (bytes) to pass through the channel of bandwidth B_1 (bytes/s) in one hop. H equals the distance in the number of hops,

Table 1. Architecture overviews.

Features	iPSC/1	iPSC/2	Ncube/10	Ametek 2010	Topology 1000
Node CPU	Intel 286	Intel 386	Custom 32-bit	Motorola 68020	Inmos T800
Clock rate (MHz)	8	16	7	25	20
Node operating system	Axis 3.0	NX/2	Axis 2.3	R Kernel	Logix OS
Node memory (bytes)	512K	Up to 16M	128K	8M	Up to 16M
Data rate (Mbytes/s)	1.25	4	0.875	20	5

and we can view T_{d1} as the routing delay of each node.

Kermani and Kleinrock[14] and Athas and Seitz[15] called the basic routing model used in second-generation multicomputers (for example, the iPSC/2 and Ametek 2010) wormhole routing. Instead of storing a packet completely in a node and then transmitting it to the next node, wormhole routing operates by advancing the head of a packet directly from incoming to outgoing channels. Only a few flow-control digits are buffered at each node. These digits, or flits, are the smallest units of information that a queue or channel can accept or refuse. A message consists of a sequence of flits, in which the flit at the head of the message governs the route, and the remaining flits follow in pipeline fashion. Besides avoiding the use of storage bandwidth in the nodes through which messages are routed, wormhole routing and its flow control also reduce the message latency caused by distance in the network. Therefore, the data transfer rate becomes the limiting factor for message-passing speed.

We can express the communication latency of the wormhole model as:

$$T_{lat2} = T_{d2}H + (K/B_2) \qquad (2)$$

where $T_{d2} = K_b/B_2$ is the routing delay in each node for sending the packet head in K_b (bytes) to pass through the channels of bandwidth B (bytes/s). K/B_2 is the time required to transmit the whole packet K (bytes) continuously through the wormhole channels of bandwidth B_2 (bytes/s), and H is the communication distance. The ratio between Equations 1 and 2 is a quantitative comparison of the two models:

$$R = \frac{T_{lat1}}{T_{lat2}} = \left(\frac{B_2}{B_1}\right) \frac{K\,H}{K_bH + K} \qquad (3)$$

The size of the packet head K_b is trivial in comparison with the size of the whole packet K. For example, the packet head size in the Ametek 2010 is only 2 bytes. Therefore the ratio R in Equation 3 may be expressed as:

$$R \approx (B_2/B_1)H \qquad (4)$$

This equation indicates that the wormhole model reduces the communication latency up to $B_2/B_1 \times H$ times over the store-and-forward model. In this case, we assumed the message size K and the communication distance H to be the same in both communication architectures, and the bandwidth of the second-generation multicomputer B_2 to be higher than the one of the first-generation B_1.

Even if the data bandwidth of the two models were the same, $B_1 = B_2$, the communication latency would be reduced to H times in the wormhole model. For example, the first-generation hypercube Intel iPSC/1 uses the store-and-forward model; its data bandwidth is 1.25 Mbytes/s. The second-generation hypercube Intel iPSC/2 uses the wormhole model; its data bandwidth is 4 Mbytes/s. If we substitute $B_1 = 1.25$, $B_2 = 4$, and H = 5 for a 32-node hypercube in Equation 4, we obtain $R \approx 16$. This ratio indicates that a 32-node iPSC/2 hypercube may reduce the communication latency time up to 16 times over a 32-node iPSC/1 hypercube.

Hardware implementation. The communication mechanisms based on the store-and-forward technique used in the Intel iPSC/1 and Ncube/10 are typical first-generation message-passing protocols on a distributed-memory multiprocessing system. The processor on each node in that path participates in handling communications, stopping other processing tasks during message-passing periods.

The iPSC/1 and the Ncube/10 consume the local memory bandwidth and computing cycles in the routing nodes while accumulating a latency of several hundred microseconds per hop. Thus, the computing speed and bandwidth in each processor mainly determine the store-and-forward communication speed. The higher the clock rate of each processor, the lower the latency in communication will be, since the processor more speedily accomplishes the store-and-forward operation. The experiment's results discussed in the next section show the low efficiency of the store-and-forward techniques on the Intel iPSC/1 and Ncube/10.

We can implement the wormhole model differently on a multicomputer. The hardware structures on the iPSC/2 and Ametek 2010 are two typical implementations for the wormhole routing model on a interconnecting network.

The wormhole routing model greatly reduces communication latency.

The direct-connect router, a hardware-controlled message-passing system in the Intel iPSC/2, forms the basis of the communication system. Think of the router as a switching network. When one node wants to communicate with another, the sending node closes a series of switches and establishes the communication path. Then, messages proceed at the full hardware speed of 4 Mbytes/s. Only the sending and destination processors participate in the communication; the other processors in the routing path continue with their normal activities. Since it takes only a few microseconds per node to build the path, the additional overhead for multihop communications is insignificant. In addition, the hardware routes messages independently, and the iPSC/2 communication latency is significantly reduced over that in the iPSC/1.

The Ametek 2010 communication network is the most efficient one among the five multiprocessing architectures. The message network consists of a 2D grid of custom mesh routing chips. Message packets advance directly from one of these chips to another in a blocking variant of cut-through routing of the wormhole routing. At the 20-MHz rate, the 8-bit-wide channels yield a communication bandwidth of 20 Mbytes/s per channel. Thus, the network quickly establishes a connection circuit between two remote nodes, and the mesh routing chips transfer messages in a byte-serial fashion in one operation.

The Topology 1000 implements the store-and-forward technique differently. The communication system is tied into each transputer at a very low level. The transputer employs a hardware scheduler to schedule the communication of messages. Therefore, setting up a communication takes just a few microseconds.

On the other hand, the transputer implements synchronized message passing. Both sender and receiver must be ready before a communication can take place. This coordination occurs at the lowest level of the communication protocol and results in the absence of problems with data overruns or buffer overflows. In addition, the store operation acts the same as it does in the iPSC/1 and Ncube/10, storing the message in the local memory of the routing node. However, each processor is only responsible for initiating the forward operation. Then the DMA data link carries out the message transfer without further interruption of the processor.

The DMA data links on the Topology 1000 operate at a maximum unidirectional rate of 1.75 Mbytes/s or a bidirectional rate of 2.5 Mbytes/s. Four links per transputer produce a 10-Mbyte/s rate. The basic idea of this model is to use *excess* parallelism to hide the latency in the data transfer. For very short messages, a low transfer rate is possible because most of the time spent in the communication occurs in the processor cycles upon initializing a data transfer. However, the communication can take advantage of a large message transfer when the processor's initialization time is trivial (compared with the data transfer time used by the DMA data links).

Experimental results on a Topology 1000 with the DMA data links show improvement in communication efficiency. The communication speed of the Topology 1000 is much higher than on the Intel iPSC/1 and Ncube/10, although all three multiprocessing systems use general store-and-forward techniques.

Comparing the two topologies. The Ncube and both iPSC systems use the hypercube interconnection topology. The Ametek 2010 uses a 2D grid as the interprocessor connecting topology. We can compare these two network topologies in terms of the communication efficiency.

We can make a hypercube of arbitrary dimension by using a linear arrangement with connecting wires. We obtain the cube of each dimension by replicating the one in the next-lower dimension and then connecting corresponding nodes. For example, directly connecting two nodes labeled 0 and 1 between the two nodes gives us a one-dimensional hypercube (2^1). We make a 2D hypercube by duplicating the bisection, or the 1D hypercube, by directly connecting the corresponding node of each bisection together. Adding a high-order bit to the node number sets it to 0 for the lower order bisection and 1 for the other. We construct the higher dimensional hypercube by further connecting the bisections of the hypercube. As Figure 1a shows, each processor in a hypercube connects to all other processors whose binary tags differ by exactly one bit. We can make a hypercube of arbitrary dimension by using a linear arrangement with connecting wires, as shown in Figure 1b.

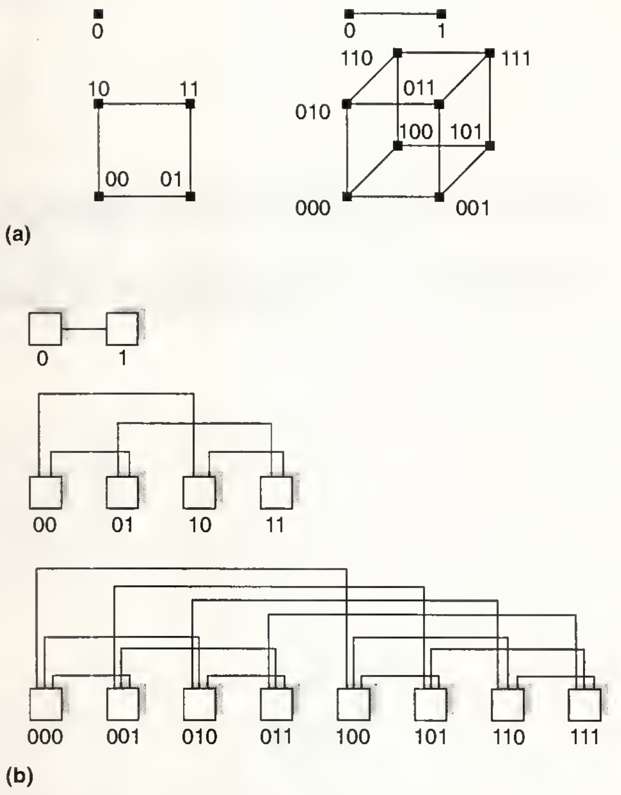


Figure 1. Construction of a hypercube.

We can make a channel that physically links two directly connected nodes from a bundle of wires consisting of data bits and any necessary control bits. We need $N/2$ channels across the bisection to construct a hypercube, where N is number of nodes in the hypercube. However, using the same method to construct a 2D grid requires $O\sqrt{N}$ channels across the bisection, where N is the number of nodes in the 2D grid. We can determine the maximum distance between a pair of

continued on p. 52

RST Cache Memory Design for a Tightly Coupled Multiprocessor System

The contention for accessing the shared memory by several processors restricts the performance of a tightly coupled multiprocessor system. Cache memories reduce this potential performance impact. Our approach involves the implementation of a coherence protocol and the cache-memory architecture for a Clipper-based multiprocessor prototype.

Cosimo A. Prete

University of Pisa

Decreased costs of certain components (RAM and CPU) and the increased demand for high-performance systems have led computer companies to develop multiprocessor systems. Among all the ideas proposed, the tightly coupled multiprocessor system[1-5] provides an appropriate solution to the demand for additional computing power. Such a system consists of a set of processors that can access shared memory and I/O devices via a common bus.

In these systems, the processing tasks are distributed among several processors working in parallel. However, achieving speed up is only possible if no bottlenecks arise. The common bus and the shared memory impose the two greatest limitations in the organization of these multiprocessors:

- When n processors want to use the common bus to access the shared memory at the same time, $n-1$ processors have to queue, which wastes time in waiting.
- Access time of the shared memory is considerably longer than the time needed by a processor to transfer the data. As a result, the processor must wait until the shared memory completes the operation.

To improve system performance under these conditions, computer designers place a small, high-speed buffer (cache memory[6]) between the common bus and each processor. Cache memory temporarily holds the data and the instructions most recently used. The success of cache memory relates to the property of locality.[7] Programs spend most of their time repetitively executing a few tight loops of code. Access to that code and to the relative data is faster if the cache memory holds these instructions and data.

In multiprocessor systems, cache memories produce two successful results: reduction of the average access time to the shared memory and minimization of bus requirements for each processor.[8] The main problem to be solved in multiprocessor systems is ensuring the consistency of the copies of the memory blocks (sequences of bytes in the shared memory) in the different caches.[9-15]

In particular, it is necessary to ensure that whenever one processor carries out a write to the memory location m, all subsequent reads of location m by any processor will deliver the new contents of m. A common approach calls for each cache to maintain the consistency of the copies. A coherence protocol specifies the cache behavior during both processor operations and bus transactions (a one- or multiple-word transfer on the common bus).

We decided to base the coherence protocol and the cache memory architecture and design for a multiprocessor system on Intergraph's (for-

0272-1732/91/0400-0016$01.00 © 1991 IEEE

merly Fairchild's) Clipper microprocessor.[16-17] We chose the Clipper for its high-performance features: fast clock speed, internal caches, internal dual buses, sophisticated pipelining system, and integrated execution units (see box). We used a private cache for each processor based on the experience of a previous project[4] in which the common bus caused the main performance bottleneck.

The coherence protocol, called Reduced State Transitions, is a modification of the well-known Dragon protocol.[18] We modified it to improve system performance. In particular, the high performance of RST results from sophisticated architectural solutions (such as the use of buffers and overlapping a processor operation and a bus transaction). Additional performance stems from the balance between several cache factors and careful tuning obtained by means of a simulation phase.

Features of the Clipper microprocessor

The Clipper[16, 17] is an advanced, 32-bit microprocessor originally developed by the Fairchild Semiconductor Corporation and now produced by Intergraph Corp. At present, this microprocessor is available on a module consisting of a processor chip, a 33.3-MHz clock controller, and two cache and memory management unit chips (see Figure A).

The processor can address two different memory spaces (user and supervisor) of 2^{32} bytes. It does this by using nine types of addressing modes by using the program counter register, a general register, and or a displacement of 12, 16, or 32 bits. Clipper instructions contain zero, one, or two operands, but only one of them resides in memory (load/store architecture). The instructions are 2, 4, 6, or 8 bytes long, with the first two bytes containing the operation code of the instruction, format type, and addressing mode. The instructions execute operations on data of 1, 2, 4, or 8 bytes, and on floating-point data at single (4-byte) and double (8-byte) precision.

The CPU contains a number of control and status registers: sixteen 32-bit general registers for supervisor mode, sixteen 32-bit general registers for user mode, and eight 64-bit floating-point registers. Two supervisor instructions can execute the data transfer between a user and a supervisor register. The processor includes a floating-point unit (FPU) conforming to the IEEE 754 Standard for Binary Floating-Point Arithmetic.[19]

The processor is connected via two 32-bit buses to two CAMMUs; one references the data, the other references the code. Each CAMMU contains a translation look-aside buffer and a 4-Kbyte, two-way, set-associative cache memory with write-through, copy-back, and noncacheable caching policies on a per-page basis. The data in the caches are organized in 128 sets, each one containing two blocks of 16 data bytes. Each TLB holds 128 of the most recently used values belonging to the translation tables for instructions and data.

The Clipper divides the virtual address space into 4-Kbyte pages. The real address space is similarly divided into 4-Kbyte pages. When mapping is enabled, the CAMMU translates virtual addresses generated by the CPU into real addresses. A two-level hierarchy of the page tables achieves the translation process. The CAMMUs interface both the CPU and FPU to the main memory and I/O devices via the Clipper bus. The Clipper high-speed, synchronous bus contains 73 signal lines extending from the Clipper module via a 96-pin connector.

Finally, the Clipper accepts 402 interrupts: 18 hardware traps (including page fault, memory error, overflow, and privileged instruction violation by a user-mode program), 128 supervisor calls, and 256 vectored interrupts.

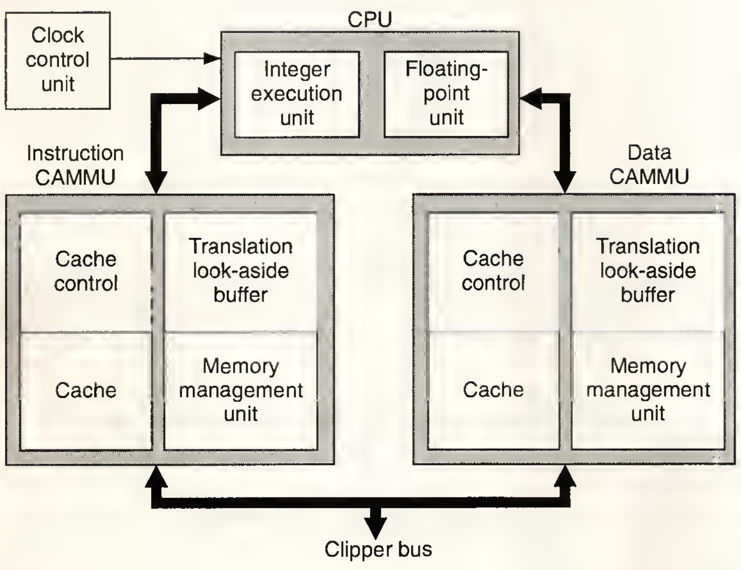


Figure A. Clipper module block diagram.

Multiprocessor cache requirements

In systems that include cache memories, every write operation requires the updating of both the copy within the cache and the memory block involved. The latter is updated either by the write-through or copy-back technique. The write-through technique updates the memory block immediately; the copy-back technique updates the memory block when the copy of another memory block replaces a modified copy in the cache (replacement phase).

At first, copy-back seems less advantageous since updating the memory block requires the transfer of an equal number of bytes to the block size, whereas write-through only requires the transfer of the modified location into memory. Write-through, however, needs a bus transaction upon each write operation. When considering a program modifies the same variable many times, we can deduce that the bus traffic produced by the copy-back is less than that produced by the write-through.

In a multiprocessor system, different caches can hold copies of the same memory block. Therefore, a strategy must be adopted to guarantee the consistency of cached copies. Multiple copies may result from

- sharing the process code,
- cooperation and communication among processes,
- execution of system primitives or I/O interrupt routines on different processors, and
- process migration from one processor to another.

In some systems, we employ a software approach that prevents multiple copies. The approach labels memory segments of shared data as noncacheable[20] and flushes all the possible modified copies of private data upon each process context switch. Actually, this solution uses the cache memory only for the private memory segments. A more fruitful approach involves a bus-watching scheme, which maintains consistency by updating the possible copy in the cache whenever a bus transaction involves the pertinent memory block.[10, 15, 21, 22]

The two memory updating techniques described previously provide the basis for the coherence protocols. In the case of protocols based on the write-through technique, each cache verifies whether it has a copy of the memory block involved in a write transaction. If so, the cache updates the copy (see Figure 1). Write-through models provide the best balance of performance and economy.[23] Copy-back models are more complex, but they make it possible to minimize the number of write transactions needed by each processor.

Typically, copy-back protocols distinguish between private copies (only one cache contains a copy of a given memory block) and shared copies (several copies of a given memory block in the caches). For a write operation on a private copy, the cache updates the copy and labels it as dirty. (For definitions of dirty and clean states, see later section on copy states). For a write operation on a shared copy, the cache updates its copy and broadcasts the write operation on the bus to update or invalidate all the possible copies.

In practical terms, the cache operates in a copy-back mode for private copy and in a write-through mode for shared copies. To implement this advanced model of coherence protocol (known as a distributed write model or broadcast write model), we need to

- dynamically evaluate the kind of copy (private or shared). We cannot make this copy discrimination based on the kind of segment (private or shared) to which the memory block belongs. Some events, such as process migration, create shared copies from memory blocks belonging to private areas of processes;
- maintain the type of each cached copy; and
- permit a cache, with a dirty copy of a memory block involved in a read block transaction, to supply the up-to-date data.

We can usually satisfy the first requirement by adding a shared line to the common bus. During a bus transaction, each cache (apart from that belonging to the requesting processor) with a copy of the memory block activates the shared line. (In subsequent uses, we refer to the cache belonging to the requesting processor as the requesting cache, and all other caches as listening caches.) Therefore, the requesting cache considers the copy involved as shared if and only if the shared line is active; otherwise, it is considered as private.

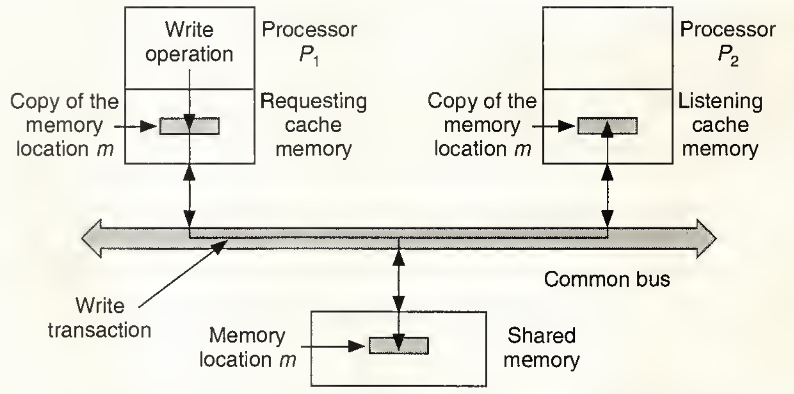


Figure 1. Consistency problem of multiple copies of the same memory block.

A traditional tagged-memory scheme[24, 25] meets the second requirement of maintaining the type of each cached copy. The scheme allows us to associate a state to each cached copy. Finally, the third requirement makes it necessary to design the cache memory as a dual-port memory unit. High-performance requirements demand a cache design that permits overlapping between the activity pertinent to processor operations and the bus-watching activity. These concurrent activities need duplication of the tagged-memory scheme and the use of arbiters for internally shared structures to detect and avoid particular conditions that provoke inconsistencies. Solutions for all the cache problems involved in the RST design appear in a later section.

RST coherence protocol

To describe this coherence protocol, we list the possible copy states and the cache actions performed upon a local processor operation or bus transaction. We assume that the common bus includes the shared line used by all the caches as described previously.

Copy states. Each cache block keeps a copy of a memory block in one of the following states:

- *Private and clean.* Only one copy of the memory block exists and it is identical to the memory block.
- *Private and dirty.* Only one copy of the memory block exists, but it is not identical to the memory block. When another cache requires a copy of this memory block, the cache with the copy must substitute for the shared memory by furnishing the copy itself. When such a copy is selected for replacement, it has to be written back to shared memory.
- *Shared and clean.* Several copies of the memory block may exist in as many caches.
- *Shared and dirty.* Several copies of the memory block may exist in as many caches. These copies (one shared and dirty and the rest shared and clean) are identical to one another but different from the memory block. When another cache requires a copy of the same memory block, the cache holding the shared and dirty copy must substitute for the shared memory by furnishing the copy. (In this way, a private and dirty or a shared and dirty copy takes the place of the memory block.) When such a copy is selected for replacement, it must be written back to shared memory.

Local processor operations. The cache behavior upon a memory operation provided by the local processor depends on the type of operation (read or write) and whether the cache has a copy of the involved memory block (hit or miss condition).

Read hit. When a cache hit occurs during a read operation, the cache directly supplies the contents of the location to the processor without any further actions (see Figure 2).

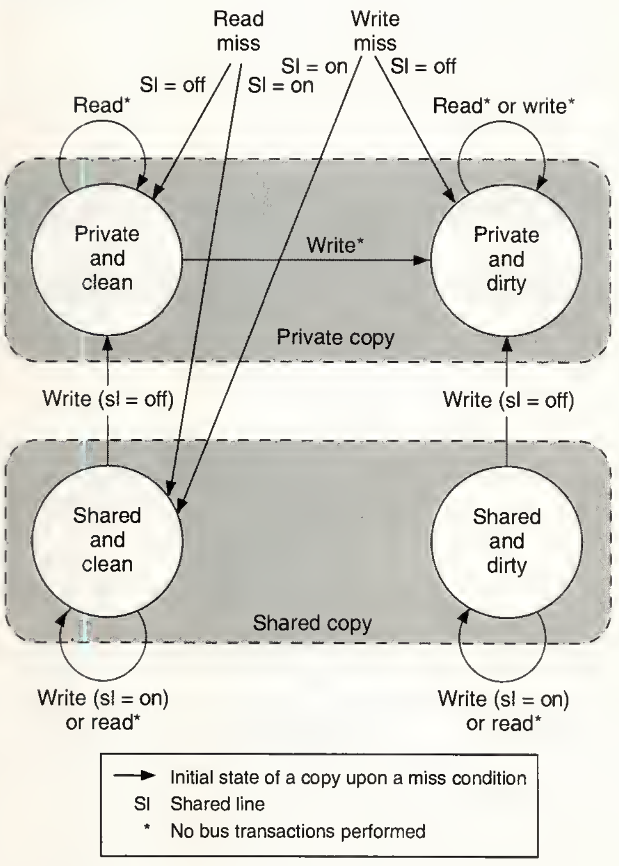


Figure 2. State transitions of a cached copy involved in a processor operation.

Write hit. When a cache hit occurs during a write operation and the copy is private and clean, the cache updates the copy and changes the copy state to private and dirty. These cache actions must be indivisible to avoid the possibility that concurrent read block transaction of the same memory block could produce an inconsistent copy. The cache can achieve the indivisibleness of these actions by keeping the common bus busy while executing these actions. When the copy referenced is in the private and dirty state, the cache updates the copy and does not change its state.

Finally, when the copy is in either shared state, the cache also broadcasts the write operation on the common bus (by means of a write transaction) to update both the memory block and the other copies. As a consequence of this bus transaction, if the memory block is no longer shared (that is, the shared line is not active), the copy state changes into one of the two private states. In particular, when the copy is shared

continued on p. 40

Implementing a DSP-Based Petri-Net Simulation Tool

The proposed efficient simulator of petri nets, suitable for both loosely and tightly coupled parallel systems, features high-execution speed. We obtain this speed through an optimized software structure on an original hardware architecture, based on digital signal processors of the TMS320 family. The PN class outlined compactly represents a wide range of processes by restricting the model size and the relevant simulation times.

Antonella Di Stefano

Orazio Mirabella

Fabio Presente

University of Catania

Petri nets (PNs) represent a widely used formalism for modeling the behavior of parallel systems (see box on page 61). Present complex parallel systems (both loosely and tightly coupled) need automatic tools for system specification, implementation, verification, and evaluation.

A simulator is a basic module for such tools. It supports the designer in tracing the evolution of the system under study, verifying its correctness (such as the absence of deadlock),[1,2] and evaluating its performance.[3-6]

In addition, a simulator can directly implement tightly coupled parallel systems for process control purposes[7] when the simulator's evolution time is real and compatible with the system time requirements. This activity is conditional because of the existence of a suitable I/O interface with the controlled process.

We suggest using an efficient PN simulator suitable for both loosely and tightly coupled parallel systems that features high-execution speed. We obtain this efficiency through an optimized software structure on an original hardware architecture based on digital signal processors from the TMS320 family.

Essential items in the development of a PN simulator are the:

- choice of a suitable PN class (timed or stochastic, with or without inhibitor arcs, and so on) for modeling the system to be simulated,
- data structure that allows the user to represent the PN on a computer memory,
- iterating algorithm that simulates the PN evolution, and
- hardware support for efficient implementation of the previous items.

PN simulation algorithm

The framework of the proposed PN simulator consists of two main steps, configuration and PN evolution.

Configuration. The PN is represented in the system memory according to a suitable schema for subsequent analysis of the possible PN evolution. The place marking gives the PN state at any time. PN behavior continues at any time from the knowledge of the input places (preconditions, or pre), output places (postconditions, or post), firing time, time-out, and any procedure associated with each transition. PN representation must be compact and efficiently handled by the enabling/firing rules to minimize the processing time and the amount of data stored.

PM matrix rows store the transitions, which are numbered in a strictly sequential layout. Each row matches the relevant transition with the same layout number to simplify and to speed up its location. According to the items in Table 1, a row is divided into twelve 16-bit fields (see Figure 1 and Tables 2 and 3).

0272-1732/91/0400-0020$01.00 © 1991 IEEE

Table 1. Divisions of PM matrix rows.

Fields	Comments
1-5	Identifies the transition's input places
6-10	Identifies the transition's output places
11	Splits into two bytes: The lower byte lists the procedure number (if any). The upper byte contains two different pieces of information: the conflict bit (the most significant bit) and the status flag (the six lowest bits)
12	Identifies the time associated with the transition

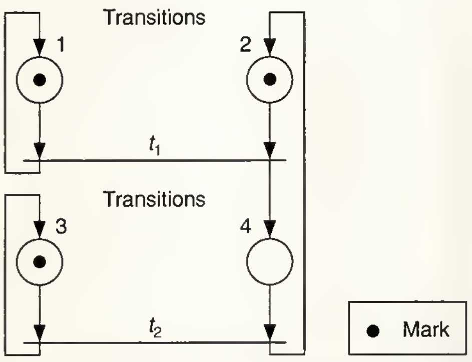


Figure 1. A sample PN used as reference to explain the matrices P and PM.

Table 2. Values in the matrix (see Figure 1).*

Transition	Input places	Output places	—	Time
1	1 2 0 0 0	1 4 0 0 0	1200	5
2	3 4 0 0 0	2 3 0 0 0	1100	100

* Values of field 11 are indicated in terms of nibbles

Table 3. Detail of the upper byte of field 11.

Conflict	Not Used	Required	Present
0	0	0 1 0	0 1 0
0	0	0 1 0	0 0 1

We point out that a status flag provides a way to quickly determine whether any transition has gained the firing configuration. In fact, the upper three bits show (Table 3) the quantity of input places to be marked to enable the transition. The lower three show the quantity of input places actually marked. In this way, a transition occurs when these bit terns are equal to each other.

The P matrix rows store the places; they are numbered in a strictly sequential layout. Each row segments into five 16-bit fields containing those transitions that hold the place as an input. In addition, the most significant bit of the first field stores information on marking occurrence (see Table 4). The places bearing the lowest layout numbers match the stop_time-out places for quickly detecting which transitions the time-out can stop. A time-out vector stores the associated start_time-out places. In this way, the P matrix permits the linking of place marking with the relevant activated transitions.

PN evolution. This evolution occurs through iterative use of the enabling/firing rules. The transition $t \in TR$ to be processed is identified on the grounds of the preconditions. Then, after a time $T(t)$, the postconditions execute the associated procedure (if any) and state the output marking.

The algorithm simulating PN evolution alternatively commutes between two phases: identification of the enabled transitions and firing of the already enabled transitions according to the time associated with each of them.

Sequential scrolling of the PM matrix (which relates to the status flag) is not a good way to detect the transitions to be enabled. In fact, such an approach would involve a rather long time period proportional to the PN dimensions. The current approach detects transitions after any marking of new places, which are postconditions of the just fired transitions.

At each new place marking, the status flag is checked relevant to those transitions holding that place as an input. The P matrix directly accesses these transitions. Through a transition *SF* in the PM matrix it is possible to recognize whether the firing configuration has been gained. Therefore, we can quickly recognize the transitions by simply checking only those PN elements modified by the firing of one or more transitions without considering the remaining part of the net.

Table 4. Values in matrix P for the sample PN.

Place	Transitions
1	1 0 0 0 0
2	1 0 0 0 0
3	2 0 0 0 0
4	2 0 0 0 0

The transition number that represents the transition identifier is put into list E. We only need the whole PM matrix list at the start of the simulation session to detect the transition kernels initially suitable for insertion into list E.

After any change in the marking configuration, the simulation process pulls transitions (one or more) with the shortest time-out of list E and fires, thus determining a new marking configuration. The simulation time counter is contemporaneously increased by the quantity of time associated with the fired transition(s). Therefore, the simulation time, in such an approach, is not real time: it only represents the PN evolution time.

Figure 2 shows a flowchart of our simulator algorithm.

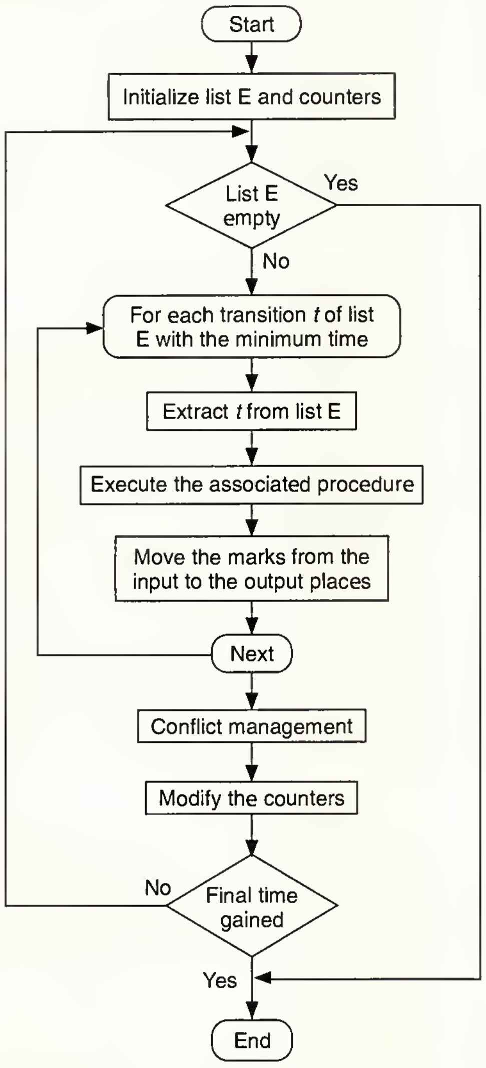

Start
Initialize list E and counters
List E empty
Yes
No
For each transition *t* of list E with the minimum time
Extract *t* from list E
Execute the associated procedure
Move the marks from the input to the output places
Next
Conflict management
Modify the counters
No
Final time gained
Yes
End

Figure 2. Flowchart of the simulator algorithm.

Conflict management. We must generate a new list C to define the management of two or more conflicting transitions. The handling of the list increases the processing time as compared with that needed by the basic module.

The conflict in progress produces a nondeterministic choice of the transition to be enabled among those in conflict. Obviously conflict solution is an intermediate step prior to updating list E. The subsidiary list C acts as a filter, allowing disposal of all the transitions that cannot occur. The simulation program quickly identifies all the transitions offering a potential conflict by checking the conflict bit.

At any configuration change, the simulation process inserts all transitions with a complete input marking—which could promote a conflict—into C. List E, on the other hand, directly receives all the enabled transitions without potential conflict.

Of those remaining in C, only a single transition per time inserts into E according to a nondeterministic selection. This process reiterates until C is empty. During conflict management, a comparison of lists C and E occurs each time a transition moves from C to E. All transitions in C in conflict with transitions in E are taken out of C. This approach allows us to select only one transition (to be enabled) among those in conflict and to discard the others.

Figure 3 contains a flowchart of conflict management.

The simple example (see Figure B in box on page 61) clarifies the mechanism adopted for conflict management. We suppose at the time t only places P_1 and P_2 are marked; also, at the time $t' > t$ P_3 and P_4 are also marked. Transitions T_1, T_2, T_3, and T_4 conflict and are thus inserted into list C. According to the nondeterministic choice, the conflict management algorithm determines several situations:

- selection and insertion of T_2 into list E: T_2 and T_3 still conflict with T_2; list C discards them. T_4 no longer conflicts with another transition and moves into E. (The same applies if the algorithm selects T_1.);
- selection and insertion of T_3 into E: T_1, T_2, and T_4 all conflict with T_3; C discards them;
- selection and insertion of T_4 into E: C discards T_3, whereas T_1 and T_2 remain in C for a further selection process (only one of the two transitions T_1 and T_2 will enable).

Time-out management. We handle a time-out transition as a couple of transitions: the start_time-out transition with an start_time-out input place and a stop_time-out transition with a stop_time-out input place. Of course these transitions hold different output places.

The time-out check performs at each configuration change. A suitable memory location, called time-out management, stores the quantity of stop_time-out places at the beginning of the simulation session.

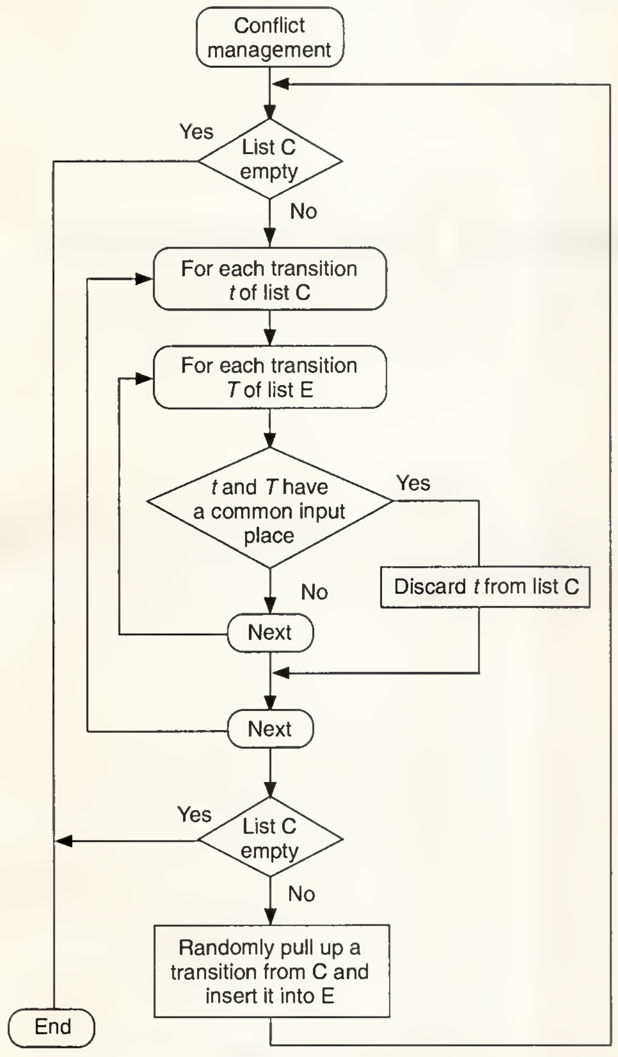

Conflict management
List C empty
Yes
No
For each transition t of list C
For each transition T of list E
t and T have a common input place
Yes
No
Discard t from list C
Next
Next
List C empty
Yes
No
Randomly pull up a transition from C and insert it into E
End

Figure 3. Flowchart of conflict management.

The places numbered from 1 to time-out management in the P matrix are stop_time-out places. Thus, the marking of each new place during PN evolution involves checking out its sequential number to test whether is a stop_time-out place (that is, whether the place number is lower than or equal to time-out management). In the latter case, checking the time-out vector tests whether the corresponding start_time-out place has enabled the corresponding transition.

When a stop_time-out place marking occurs while the start_time-out transition is still in E, the identification and removal of the stop_time-out place occurs. Then the stopping transition moves into list E.

When the start_time-out place marking has not yet enabled the time-out mechanism, the marking of the stop_time-out place is ignored, according to the meaning credited to the time-out transition.

Procedure. The simulator detects whether a procedure is associated with the transition associate by checking the procedure field of the PM matrix. Testing of a procedure field occurs each time a transition is pulled out of list E. The control allows the simulator to identify the procedure, if any, through its sequential number, then through the procedure performed.

Simulator implementation

The simulator we outline here presents two features for suitable hardware support and efficient implementation: the capability to quickly access its memory, and actual and quick handling of its pointers.

These features permit the easy handling of the data representing the PN, as well as fast repetition of the simulation steps throughout the evolution of the PN.

Many microprocessors satisfy such requirements with different powers. Among them, the DSPs of the TMS320 family show a good level of performance. This family uses a modified Harvard architecture that offers overlapping between the fetch and execute phases of an instruction. At the same time, the architecture allows the user to exchange information between data and program areas.

Some of the most interesting features of these processors, which make them suitable for the proposed application, are:

- the capability to perform almost all the instructions in only one machine cycle, which determines a high throughput (close to 5 million instructions per second);
- a 32-bit multiplier that performs 16 × 16-bit multiplication in only one machine cycle;
- an on-chip memory of different size depending on the device adopted (such as five-hundred-forty-four 16-bit words on TMS32020) for quick data storage or access; and
- a secondary ALU for handling some registers in parallel with the main ALU (on the TMS32020 or newer chips).

These features perform an efficient selection of the transitions to be enabled. Since two matrices represent a PN, we widely use Compare and Multiply instructions to calculate the positions (inside the matrices) of the transitions to be enabled.

TMS320 DSPs[8,9] are also fit for general-purpose applications, in addition to the main features described earlier.

TMS32020 implementation. We used a board to implement a simulator on a TMS32020, which presents the following main components:

continued on p. 56

Three-Dimensional Optical Architecture
and Data-Parallel Algorithms for Massively Parallel Computing

The parallel nature of optics and free-space propagation, together with its freedom from communication interference, makes it ideal for designing massively parallel computers. Our architecture is highly amenable to optical implementations and aims at data-parallel applications.

Ahmed Louri

University of Arizona

Optics, due to its inherent parallelism and noninterfering communications, is under serious consideration for designs of massively parallel processing systems of the future. To contribute to this undertaking, designers at the University of Arizona's Department of Electrical and Computing Engineering explored a three-dimensional optical computing architecture under a grant from the US National Science Foundation.

This model—a single-instruction, multiple-data system, or SIMD—exploits spatial parallelism and processes 2D binary images as fundamental computational entities based on symbolic substitution logic. A better alternative than electronic mesh computers, this system effectively implements highly structured data-parallel algorithms, such as signal and image processing, partial differential equations, multidimensional numerical transforms, and numerical supercomputing. The model includes a hierarchical mapping technique that helps design the algorithms and maps them onto the proposed optical architecture.

We estimated the theoretical performance of the optical system and compared it with electronic SIMD array processors. Preliminary results show that the system provides greater computational throughput and efficiency than its electronic counterparts.

Background

The tremendous progress in science and technology introduced an increase in the processing of large amounts of data in real time in a wide variety of scientific applications. For example, real-time computer vision requires processing images of $1,000 \times 1,000$ data elements within a time frame of 16.7 milliseconds. This time suggests processing rates of 10 to 1,000 GOPS (10^9 operations per second) and input data rates approaching 1 Gbyte/s.

A common factor of these applications is a high degree of data parallelism in which simple arithmetic and logic operations must simultaneously take place across all data points.[1] Computing these applications with high-throughput rates requires massively parallel processing; however, traditional electronic technology faces major limitations in achieving massive parallelism. A key feature of this type of processing is the large amount of communication required among the processing elements (PEs). While the design of high-performance PEs has progressed significantly, the progress in designing high-performance interconnection networks has not been satisfactory. The major bottlenecks in today's massively parallel processing systems include the limited communication bandwidth and the lack of cost-effective means of achieving parallel I/O.[2-4]

0272-1732/91/0400-0024$01.00 © 1991 IEEE

Several researchers[5-9] suggest optics as a complementary technology for breaking major performance barriers faced by conventional electronic technology. Optics has many unique features that can be exploited for high-speed parallel processing. They include speed, parallelism, adequate communications, and architectural flexibility.

Optical systems are inherently multidimensional. Lenses, prisms, and mirrors can transfer planes comprising over a million data points simultaneously. This fact implies that a cost-effective parallel means of achieving I/O and multidimensional architectural topologies may be possible. The rate at which data moves through an optical processing system is essentially limited by the rate at which data enters the system and its detection at the output. The actual computation time consists mainly of light propagation through optical devices (provided that the switching rates of these active devices are comparable to optical signal propagation). Thus, we can obtain higher throughput and processing rates than we do with current systems.

Perhaps the most attractive feature of optics for massively parallel processing is communications.[3,10,11] Transmission of information via photons requires no physical conducting material, but relies on low-loss dielectric material for waveguide propagation or free space. As a result, optics-based interconnections potentially offer a freedom from mutual effects not afforded by electronic interconnections. This advantage becomes more important as the bandwidth of the interconnections increases, for the effect of mutual coupling associated with electrical interconnections is proportional to the frequency of the signals propagating on the interconnect lines. Therefore, optics-based communications offers higher temporal and spatial bandwidths.

The noninterfering nature of optical interconnections offers extra flexibility in routing, which in turn offers more architectural flexibility. Since electrical interconnections cannot cross, they must be routed under one another. Optical interconnections can cross one another without negative effects. Moreover, since optics-based interconnections require no mechanical contacts, we simply change the directions of optical beams to reroute interconnections. Various sources provide more details on optical interconnections.[10-14]

While the justifications for using optics for interconnections as well as mass storage are well established, the justification for using optics in digital processing remains in an embryonic stage, since digital optical device developments are in their infancy. However, if data must be converted to optical form to use an optics-based communication medium, using an optical computing engine might keep up with the rate of communications, without resorting to signal conversions (electronic-optical-electronic). These conversions cause major performance degradation and increase power consumption.

The possibility of using optics for building new parallel computing systems tailored to the requirements of data-intensive applications has been an objective of several researchers. Recent technological advances in optical devices raised hopes for the practical realization of new parallel optical computers. These advances include the development of compounds in multiple quantum wells[15] for high-efficiency injection lasers, the development of nonlinear materials for optical switching devices,[16-20] and the development of optical logic devices capable of implementing logic functions and serving as memory storage.[21-24]

The 3D optical architecture

The driving features of optical systems—the massive fine-grain parallelism and the high degree of communication flexibility—and our ability to move around large optical images of bright and dark spots with great ease using optical components suit many applications. Such applications require the processing of large amounts of structured data (multidimensional arrays) and favor the SIMD mode of computation. The attractiveness of these attributes is evidenced by the number of SIMD optical architectures that have been proposed in the past.[25-40] The optical model we present also exploits optics advantages for parallel processing.

The 3D optical architecture. Figure 1 depicts a block diagram of the basic components of the optical architecture. Unlike conventional computers that manipulate individual 0s and 1s as basic computational objects, the optical architecture

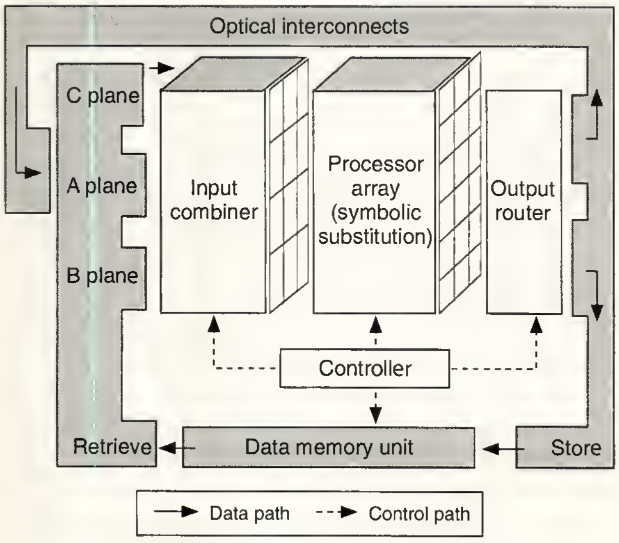


Figure 1. A schematic diagram of a 3D optical architecture for massively parallel computing.

manipulates bit planes as basic computational entities. Each bit plane i corresponds to a weight factor 2^i in the binary representation, and up to three bit planes can be processed simultaneously. For images of $n \times n$ elements, up to $3n^2$ operations process concurrently.

The heart of the architecture is the parallel processor array. Locally, this array can be viewed as a bit-serial or a bit-slice processor, since it performs one logical operation on one, two, or three 1-bit operands. Globally, it can be viewed as a plane-parallel processor, since it simultaneously performs the same operation on a large set of operands encoded as bit planes. This bit-serial processing allows flexible data formats and almost unlimited precision. Optical interconnections move the images around the system. We conceived the architecture as being built with optical hardware that manipulates entire images simultaneously both at I/O and processing. In this way, the 2D parallelism is sustained throughout various stages of the computation.

Processor array organization. The processor array operates in the SIMD mode of computation—the same operation applies to all data entries. Processing is based on optical symbolic substitution logic, or SSL,[41] described in detail later. The processor array uses three fundamental operators: a logical Not, a logical And, and a full Add as defined below, along with some other basic terms:

- **Definition 1.** We define a bit plane as $I \times I \longrightarrow \{0,1\}$, where I is a set of integers. Hence, we denote it as $A = \{a_{ij}\}$, where i, j represents the Cartesian coordinates of the binary value $a_{ij} \in \{0,1\}$. For an $n \times n$ bit plane, $i, j = 1, \ldots, n$. We define a 0 plane as an $n \times n$ bit plane A, such that $a_{ij} = 0$ for all $i, j = 1, \ldots, n$. Similarly, we define a 1 plane as an $n \times n$ bit plane such that $a_{ij} = 1$ for all $i, j = 1, \ldots, n$.
- **Definition 2.** We define a data plane of length q as a stack of q-bit planes, denoted by the boldface notation $\mathbf{A} = A_{q-1}, A_{q-2}, \ldots, A_0$, where A_{q-1} and A_0 are the most significant and the least significant bit planes respectively. We will also denote $\mathbf{A} = \{\mathbf{a}_{ij}\}$, where $\mathbf{a}_{ij}$ is an integer number.
- **Definition 3.** We define a plane negation operator denoted by P-Not(A) as one that takes a bit plane A as input and produces an output bit plane A' as follows: P-Not(A) = A' where $A' = \{a'_{ij}\}$ for $i, j = 1, \ldots, n$.
- **Definition 4.** We define a plane logical And operator, denoted by P-And, as one that takes two bit planes A,B as arguments and produces an output bit plane X as follows: P-And(A,B) = X, such that $x_{ij} = a_{ij} \wedge b_{ij}$, where $\wedge$ is the conventional logical And applied to single bits.
- **Definition 5.** We define a plane full Add operator, denoted by P-Add, as one that adds three bit planes A,B,C and produces two output planes X and Y, defined as follows: P-Add(A,B,C) = X,Y where $x_{ij} = a_{ij} \oplus b_{ij} \oplus c_{ij}$ (sum bits) and $y_{ij} = (a_{ij} \wedge b_{ij}) \vee (a_{ij} \oplus b_{ij}) \wedge c_{ij}$ (carry bits).

The signs $\oplus$ and $\vee$ denote the conventional logical exclusive Or, and the logical Or operations respectively. The three fundamental operators constitute a complete logic and arithmetic set capable of computing any arithmetic or logic function using bit-serial algorithms.

Data-routing functions. A distinctive feature of the bit-plane architecture is that it provides parallel data movement along with the parallel processor array. We can load the binary images at the input in plane format, either from the data memory or from the external world such as a television scanner or a remote-sensing device (Figure 1 again).

The data enters the processor array through three input planes A,B,C, which are necessary for bit-serial arithmetic. Planes A and B hold the operands, while plane C holds the carry-bit plane required in bit-serial arithmetic. Depending on the primitive operator needed at a given computational step, the input combiner performs three data movement functions as elaborated next.

For the logical P-Not operator, the input combiner latches onto the relevant input plane with the data to be inverted into the processor array without any change in the spatial position of the data. The logical P-And operator is applied to two bit planes in which the logical And operation proceeds on overlapping bits from the two bit planes. The data movement function required in this case, the 2D perfect shuffle, performs the perfect shuffle[42] function on the rows of the two relevant input planes, while leaving the column positions unchanged.

Given two $n \times n$ input planes, $A = \{a_{ij}\}$ and $B = \{b_{ij}\}$, where $a_{i,j}, b_{i,j} \in \{0, 1\}$. The 2D perfect shuffle of A and B, denoted by 2D PS(A,B), results in a bit plane of size $2n \times n$ as follows:

$$\text{2D PS}(A,B) = X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ x_{2,1} & \cdots & x_{2,n} \\ x_{3,1} & \cdots & x_{3,n} \\ x_{4,1} & \cdots & x_{4,n} \\ & \vdots & \\ x_{2n-1,1} & \cdots & x_{2n-1,n} \\ x_{2n,1} & \cdots & x_{2n,n} \end{bmatrix} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ b_{1,1} & \cdots & b_{1,n} \\ a_{2,1} & \cdots & a_{2,n} \\ b_{2,1} & \cdots & b_{2,n} \\ & \vdots & \\ a_{n,1} & \cdots & a_{n,n} \\ b_{n,1} & \cdots & b_{n,n} \end{bmatrix} \quad (1)$$

The P-Add operator adds the overlapping bits of three bit planes. The permutation function required for the data, the 2D 3-shuffle, is similar in function to the 2D perfect shuffle just described, except that the 2D 3-shuffle alternates rows of three bit planes. Given three input planes A,B,C, of size $n \times n$, the resulting 3-shuffled image D measures $3n \times n$ defined as:

2D 3-PS $(A,B,C) = D =$

$$\begin{bmatrix} d_{1,1} & \cdots d_{1,n} \\ d_{2,1} & \cdots d_{2,n} \\ d_{3,1} & \cdots d_{3,n} \\ d_{4,1} & \cdots d_{4,n} \\ d_{5,1} & \cdots d_{5,n} \\ d_{6,1} & \cdots d_{6,n} \\ \cdot & \\ \cdot & \\ \cdot & \\ d_{3n-2,1} & \cdots d_{3n-2,n} \\ d_{3n-1,1} & \cdots c_{3n-1,n} \\ d_{3n,1} & \cdots d_{3n,n} \end{bmatrix} = \begin{bmatrix} a_{1,1} \cdots a_{1,n} \\ b_{1,1} \cdots b_{1,n} \\ c_{1,1} \cdots c_{1,n} \\ a_{2,1} \cdots a_{2,n} \\ b_{2,1} \cdots b_{2,n} \\ c_{2,1} \cdots c_{2,n} \\ \cdot \\ \cdot \\ \cdot \\ a_{n,1} \cdots a_{n,n} \\ b_{n,1} \cdots b_{n,n} \\ c_{n,1} \cdots c_{n,n} \end{bmatrix} \qquad (2)$$

The shorthand expression 2D 3-PS (A,B,C) means the 2D 3-shuffle of planes A, B, and C. Note that the bits $(a_{1,1}, b_{1,1}, c_{1,1})$, $(a_{2,1}, b_{2,1}, c_{2,1})$, and so on become spatially adjacent after the 2D 3-shuffle permutation. The need for these data permutations will become clear when we present the optical implementation of the three fundamental operators.

The output router directs the processed data to its appropriate destination. It performs three data movement functions:

- feeding back to the input combiner a partial result needed in the next iteration, such as the carry bit plane resulting from a full Add operation;
- sending a final result to memory for storage; and
- shifting the processor array output in the X and Y directions by a programmable integer number of pixels.

The shifting functions enable communication between pixels. By means of this spatial shifting, data moves among widely and arbitrarily separated locations in the image. Furthermore, these shift functions add the flexibility of executing recursive data-parallel algorithms in which the same processing steps are applied to a reduced set of data at each iteration. The shifting functions considered here are logical shifts, in which rows or columns of zeros enter from the opposite direction of shift (rows of zeros for shifting along the Y axis, and columns of zeros along the X axis).

Symbolic substitution logic

SSL is an optical computing technique that was introduced[41] to take advantage of the massive parallelism and high speed in optics. In this method, optical patterns within a 2D binary image represent information. An optical pattern is a spatial arrangement of dark and bright spots corresponding to binary values 0 and 1. Using SSL, we can consider data as optical patterns, and processing as transformation rules. Computation proceeds by transforming optical patterns into other patterns according to predefined SSL rules. This computing technique is sensitive not only to the values of pixels

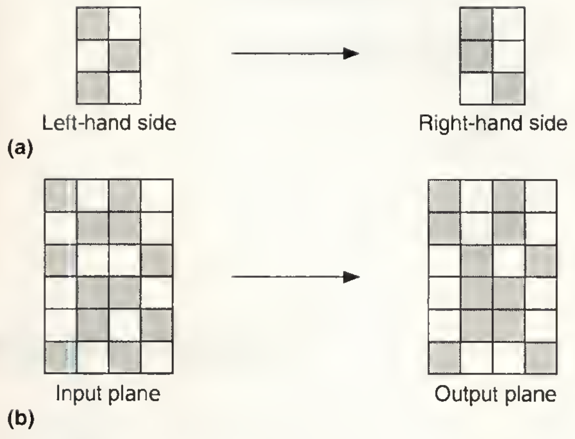


Figure 2. The concept of optical symbolic substitution logic: an example of an SSL rule (a) and its application to this input plane (b).

carrying information but also to their spatial locations in the image.

In its operation, SSL consists of two processing phases. A recognition phase detects the presence of a specific search pattern within an optical binary image. A following phase substitutes a different pattern in all locations in which the search pattern was found. Note that searching for all occurrences of the search pattern and substituting of the replacement pattern occur in parallel.

Figure 2a shows an example of an SSL rule and Figure 2b illustrates its application to a 2D image. The left-hand side pattern (search pattern) of the SSL rule is searched in the input image and then replaced by the right-hand side (replacement pattern). All locations of the search pattern are recognized in parallel.

Similarly, all replacements proceed in parallel. Since the input image can be very large (say, 1,000 × 1,000 pixels), over a million data items can be processed simultaneously. In optics, parallel search and parallel replacement of optically encoded data proceed relatively easily in parallel. Hence, an optical processor based on SSL introduces a huge amount of parallelism with little overhead of communication, data addressing, and loop indexing. Many researchers[31,44-54] have investigated optical implementations of the two processing phases of SSL.

Two-dimensional symbolic substitution rules. To implement the fundamental operators (P-Add, logical P-And, logical P-Not) optically, we need an optical property to represent the logical values 0 and 1. We can use several properties of light: intensity, polarization, and optical signal phase. One representation would encode the logic value 0 by two

continued on p. 65

Micro Review

Tracking the innovators

Richard Mateosian

2919 Forest Avenue

Berkeley, CA

94705-1310

(415) 540-7745

The fruits of fine minds

Postscript Language Reference Manual, 2nd ed., Adobe Systems, (Addison-Wesley, Reading, Mass., 1990, 772 pp., $28.95)

This is an outstanding reference work about an outstanding technical achievement. An epilogue, accurately but quaintly titled Colophon, credits Ed Taft and Jeff Walden as the authors. John Warnock and Chuck Geschke signed the preface. It's not clear who deserves credit for the exceptionally clear exposition, but Warnock and Geschke are surely responsible for the clarity and comprehensiveness of the ideas underlying Postscript. Warnock brought the initial ideas from Evans and Sutherland to Xerox's Palo Alto Research Center, where they gave rise to Interpress. Then he left PARC with Geschke to form Adobe and refine these ideas into Postscript.

Postscript addresses a hugely complex problem, namely, how to detach the physical production of graphical images from the programs that specify the images. Without Postscript every word processor and every drawing or painting program would have to include coding to deal with the special requirements of every printer, plotter, or display screen. This was the problem with the Macintosh, the first computer to support interesting graphical output. Only its Imagewriter dot-matrix printer could support the Mac's "quick draw" output. Postscript-based laser printers freed the Macintosh from this straitjacket and allowed the desktop publishing revolution to begin. At the same time, of course, Postscript-based laser printers helped to destroy the Macintosh monopoly on desktop publishing.

Postscript is a highly evolved system that has taken 15 years to reach its present state. As a result it must balance recently understood needs against backward compatibility. Thus, for example, two levels of Postscript commands are available, and some Level 1 commands are present only to accommodate old programs. Users are warned against using these commands in new programs.

Another sop to the past is that the Document Structuring Conventions (DSCs) are regarded as optional. DSCs are part of an elaborate and well-thought-out scheme to support sophisticated and highly beneficial document processing. With DSCs, documents are device independent, and systems can implement spooling, reversal of page order, resource caching, parallel printing, collation, multiple virtual pages per physical page, and routing of color or high-resolution pages to a separate printer.

Newer Postscript programs, written using Level 2 commands, can get away from the excessive reliance on implicit arguments that characterizes the Postscript embodied in the Level 1 commands. Graphics applications tend to use implicit arguments, such as pen size and shape, fill pattern, and clipping area. Postscript's Level 2 commands provide ways to group such contextual information into data structures that can be passed to graphical functions as arguments. This feature will be important to a few programmers, while the majority will be happy enough to continue their former practices.

Regardless of the baggage it carries from the past, the Postscript of today is a magnificent edifice. It is similar in structure to Forth and has echoes of Lisp. Based on a clearly conceived imaging model, Postscript contains the programming resources to enable the efficient handling of a wide variety of display devices. It also gives special attention to the handling of text, and addresses the intricacies of color separations.

0272-1732/91/0400-0028$01.00 © 1991 IEEE

This book describes the entire Postscript system. I approached it with some reluctance, expecting another dry, unreadable tome, but I was bowled over by its excellence. It is clearly written, beautifully typeset, virtually free of grammatical and typographical errors, and nicely printed on good paper. (I did find one typo, on page 627.)

Few of us will ever have to write a Postscript program, but you should read this book as a lesson in computer science. Think about the problem it addresses. Admire the solution. This is the work of fine minds.

Programmers at Work, Susan Lammers (Tempus, Redmond, Wash., 1989, 392 pp., $9.95)

Tempus has reissued a work initially published by Microsoft Press in 1986. Susan Lammers interviewed 19 programmers "who shaped the computer industry." She asked them about "timeless matters" rather than current gossip, and many of their answers are just as interesting and enlightening now as they were then.

I found some of the interviews more interesting than others. I don't really care about Apple's internal politics during the early days of the Macintosh project. On the other hand, I'm fascinated by discussions of whether simplicity is overrated as a design objective. I'm delighted that people I respect feel it's important to learn to think before specializing in computers. I'm gratified but not surprised to find that many of those interviewed have backgrounds in and affection for mathematics. I'm dismayed that so many of those interviewed feel that the ability to concentrate disappears with age.

I found things to agree with in all of these interviews, but my favorites were Butler Lampson and John Warnock. Perhaps I am only identifying with my own generation, but I think that these men showed exceptional breadth of interest and depth of thought. I found myself agreeing with them again and again.

There is no adequate way to summarize these interviews. If you are interested in systems design or in the workings of fine minds, you'll probably enjoy this book.

Mac World Expo

In January I attended the Macworld Expo in San Francisco. The wealth of fine exhibits and the huge throngs of Macintosh enthusiasts were overwhelming. I am reviewing many excellent products, but most will have to wait for subsequent columns.

Turbo Mouse ADB (Kensington Microware Limited, New York, N.Y., $169.95)

The Turbo Mouse ADB is a trackball pointing device designed to replace the mouse supplied with your Macintosh. I saw it at the Mac World Expo, and I liked it. Subsequently, I have been using one with my Macintosh SE/30. I'm not quite used to it yet, but I think I prefer it to my mouse.

The Turbo Mouse ADB is designed to sit next to your keyboard. It is approximately 5.5 by 4.75 inches in size and it slopes at about the same angle as the keyboard. The two-inch ball rotates freely in a cavity in the center of the unit. A mouse button is on each side of the ball; each is set into the sloping face of the unit. The Turbo Mouse can sit at either side of the keyboard. A dual-in-line package button setting permits the interchanging of the functions of the buttons, which accommodates left-handed users. The unit is shipped with the DIP button in the right-handed user position.

One of the Turbo Mouse buttons acts exactly like the single button on a Macintosh mouse. The other toggles between locked down and locked up positions. These buttons allow you to step through menu selections or perform dragging operations without keeping the button held down with one finger.

The Turbo Mouse also allows simultaneous pressing of both buttons to emulate a keyboard command. Three additional DIP buttons allow you to select among seven commonly used command keys, namely N, O, W, S, P, Q, and Z. The factory setting is Command Z, which selects the "undo" function for many Macintosh applications.

I like the Turbo Mouse because it is easier to use and uses less desk space than a mouse. I would like it even better if I could align it neatly next to my keyboard. Its shape is exactly right to allow me to do this, but the unfortunate placement of the connectors, both on the keyboard and on the Turbo Mouse, make this impossible. I have to offset the mouse by about a half inch, either forward or backward, depending on how I connect my cables.

Next time I'll let you know if I still like the Turbo Mouse after extended use.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 183 Medium 184 High 185

Micro Law

The *Paperback* case Part 4, What's really going on

Richard H. Stern

Law Offices of

Richard H. Stern

1300 19th Street NW,

Suite 400

Washington, DC 20036

While software publishing houses applaud the *Paperback* decision[1], programmers and developers contend that the court does not understand software technology or how the industry works, and has an anti-"techie" attitude (a bad, effete Eastern establishment attitude) to boot. They find the "let them eat cake" posture on standardization particularly hard to accept.

The *Paperback* opinion and the industry reaction to it reveal a clash in hierarchies of values. *Paperback* has all the flavor of C.P. Snow's two-culture conflict. The court's underlying attitude fundamentally contrasts with that of those in the technological community; in a nutshell, the judge is no hacker. Those on the technology side of the two-culture border will perceive, between the lines of the *Paperback* opinion, the discourse of one uninterested in, unsympathetic to, or even hostile to the values of those on the technology side—including the value of general advancement of software. A more detached viewer, however, might simply perceive two contrasting views of what generally advances software.

The *Paperback* opinion's repetitious standard-bashing best reveals not only the clash of values but also concepts concerning where the public interest lies. To the court, the effect on authors is all important; to bring about software progress, authors need to be encouraged by rewards. Talk of standardization has no bearing on how to encourage authors; to the court, standardization is just a defendant's excuse to steal rewards from authors, in a two-player game between authors and infringers.

The court's cavalier attitude about standardization in *Paperback* is a major factor in the industry reaction to the opinion that it is a case of "them against us." The opposing system of values recognizes a third player, the user; indeed, the user is so important as to be not just a third player but, by far, the principal player.

As human-factors analysis has shown, users loathe having to learn new interfaces. It is highly unlikely that any judge who was a habitual software user would deliver "let them eat cake" remarks about user-interface standardization. Rather than being consciously willing to subordinate the interests of the user community to

Highlights

Part 1 of this series described the most recent copyright decision in the screen display/user interface field, *Lotus Development Corp.* v. *Paperback Software International,* a decision of great concern to the software industry. Part 2 addressed the court's view of the case, which was that input command structure should be legally protected as a "nonliteral" aspect of the underlying computer program. Part 3 discussed the proper role of functionality aspects in a copyright infringement analysis. Particular attention was directed to conventions and de facto standards, and their role in making end users' lives easier.

In ending the series, we turn to consideration of the general implications of the *Paperback* decision.

0272-1732/91/0400-0030$01.00 © 1991 IEEE

those of copyright property, however, it is far more likely that the court was simply oblivious to those interests because they were not presented forcefully enough to the court.

Consensus on where the public interest lies will continue to be elusive. Many users believe the public interest must be identified with that of users. That is my premise, too. Software progress depends on making software acceptable to the public (that is, users), so that they will use it. I equate growth of the total market, which means wider and wider public use of software, with software progress. That in turn I equate with the progress of science and useful arts in the field. If you start with those premises, you will probably conclude that the *Paperback* decision is a setback for software progress.

Doubtless, that is not the only view possible. Moreover, users may not know what is good for them. It may be that what is good for software publishers is, in the long run, more conducive to software progress. It may be that weak protection of user-interface rights would so hamper capital formation and discourage innovation in the industry that the industry would be far worse off than if users must learn new interfaces (or stay with those products that originally set de facto standards).

There is really no way to settle that kind of argument. Still, the claims for the need of enhanced legal protection, as just described, are academic and conjectural. But the pain of learning new interfaces is clear and present to users.

What is going on here—more generally?

Review of the screen-display/user-interface decisions to date suggests that two different things are going on in these judicial decisions. The overall trend reflected in the decisions on screen displays described in the earlier Micro Law series on screen displays and user interfaces[2] is that the courts are stating reasonable general principles for decision making. But sometimes courts experience difficulties in applying the principles of copyright law to the facts. Apparently, the courts share a judicial consensus that copyright law applies in some way to screen displays, and that copyright protects only the nonfunctional aspects of screen displays. However, they have only mixed success in distinguishing the functional from the nonfunctional. The unintended result, at times, may therefore be that a court preempts an important design technique, even though the court recognized that copyright law should not be interpreted to do that.

A major problem with trying to find guidance in the case law: There are few precedents.

At least two different ways are available to determine whether a given user-interface feature is functional. One way, exemplified in the *Paperback* opinion, is for a court to begin by defining (that is, arbitrarily assuming) the task or function being accomplished. Then the court decides whether the feature in question is one of a few or many ways to accomplish the task. If the feature is one of a few or the only way to accomplish the designated task, the court holds that the feature is part of idea, functional, and thus unprotected by copyright. If the feature is one of many ways to accomplish the designated task, the court holds that the feature is part of expression, nonfunctional, and thus protected by copyright.

This kind of inquiry tends to go wrong in a number of ways. For example, the decision regarding *the* function being accomplished is arbitrary and ad hoc. Yet, that decision determines the outcome of the copyright infringement suit. A court can characterize as functional or nonfunctional any given feature of a computer program, depending on how concretely or abstractly the function of the computer program is defined. When the function of a computer program is defined at a very high level of abstraction and generality, there are always other ways of accomplishing the function; hence, function does not dictate use of the expedient in controversy. The more narrowly the function of a computer program is defined, the fewer alternative ways there are to accomplish that function; hence, function dictates use of the expedient. The same principle applies to determining function in user interfaces. Piling inference on assumption is an inherently unreliable way to reach conclusions.

The Micro Law series on legal protection of screen displays and other user interfaces suggested another way to determine whether a feature is functional. That way asks whether use of a particular feature made it easier to use the computer program associated with the user interface, faster to use it, reduced errors, or otherwise facilitated customers' use of the program or saved them (or their employers) money in using it. If the answer was affirmative, the feature was functional or utilitarian. In the main, function and utility were based on the teachings of the science of human-factors analysis and what is often termed "user friendliness." That concept of function includes *de jure* and de facto standards. It also includes convention, even when convention is arbitrary and the result of historical accident.

(It might be said, erroneously, that this approach also requires an initial determination to be made that could decide the outcome, just as in the case of the inquiry with which this one is contrasted. That is, one must ask:

Cheaper or faster to do what? The indicated parallel, however, is fallacious. The reason is that the answer to the question is: Cheaper or faster to do anything that the user wants to do with the computer program. It is not necessary to fix on *the* function and then have all conclusions follow from that determination. Any and every function is material.)

One of the things that has gone wrong in some of the cases has been that courts reached wrong-minded results by using the assumption-and-inference method of analysis instead of the easier-faster-cheaper method. We see this reflected in the mixed success of courts in response to attempts to secure legal protection of user-interface features that are dictated or suggested by human-factors analysis. These features have included techniques such as centering captions at the top of screens, using blue backgrounds, highlighting and capitalizing letters of words in menu screens that represent the keystrokes that a user must enter to invoke the command or feature associated with the word, and using particular keystrokes for particular functions.

At times, counsel and courts have taken the position that such features should be protected by the copyright in a program or screen display, despite the superior utility of such features to users, or even because of it. Other times, courts have agreed with counsel who urged the converse position—that such utilitarian features belong in the public domain.

A major problem with trying to find guidance in the case law is that there are few legal precedents. There is also a large standard deviation from the mean in the judicial decisions that exist, a great deal of saber rattling by copyright owners who are potential plantiffs, and a consequently nervous software community. Whatever guidance can be found is insufficient and unsatisfactory. Given the current state of the case law, it would be premature to declare victory or defeat for right thinking.

That these suits reach the courts, and the outcome of at least part of such litigation, illustrate the risks that legal protection of screen displays can pose, under our present legal setting. Our legal system is one in which any software publisher or lawyer is free to run any copyright theory whatever up the legal flagpole to see whether a naive tribunal will salute it. Some do.

If we have a few more* Paperbacks, *we'll have to declare the system broken.

That is another thing happening in these cases. The difficulties of applying copyright doctrines developed for books, music, and paintings to computer software, in some cases, exceed the abilities of some courts. These courts do not understand computer software technology and perhaps do not understand copyright law either.

Counsel for software publishers—determined to stamp out competitors, regardless of whether, or perhaps particularly when, the latter offer enhancements of the original software—now do more than simply try to persuade courts that functional features are arbitrary and expressive. In the true spirit of the adversary system of justice, counsel advance very expansive legal theories of the kind of subject matter protected by a copyright registration of a computer program. In the *Paperback* case, the court bit on the hook and declared command structures in user interfaces (that is, input-command languages) to be copyrighted and infringed.

The traditional legal short answer to all of those objections is that fact-finding mistakes can always be made, and appellate courts exist to correct error (plain error in fact-finding and legal error). Moreover, screen designers (or user-interface designers) accused of infringement should hire counsel who will present their cases properly. That short answer does not eliminate concern, however, by those who fear that perennially underfunded start-up entrepreneurs too often will be stifled by the threat of litigation. They also fear that plain error will too often occur when such controversies are presented for resolution to judges who are not technically sophisticated. The end result could be to hinder technological advance in this field.

Probably, two concerns of this type dominate the feelings of members of the software industry. The first is free-floating anxiety about what will happen next. Software is still in large part a cottage industry. The cottagers are not legally sophisticated industrial barons. For one reason or another, they worry that "the lawyers" are about to "do a number on them." The other concern is long-term, a concern about cumulative effect.

Some recent comments on the *Paperback* decision and its aftermath, by industry columnist Rachel Parker, illustrate the first concern.[3] Parker described Lotus's copyright infringement suit against Borland for marketing Quattro Pro, a spreadsheet much less similar to 1-2-3 than VP-Planner is, immediately after Lotus prevailed against VP-Planner in the *Paperback* case. She noted that Lotus had brought the *Paperback* case to establish the law or its limits, and said after the court's decision, "Now, we know where the law stands."

Parker must be unfamiliar with established legal principles concerning one good bite at the apple deserving another, and camels' noses pioneering the way for the rest of the camel to occupy a tent. She therefore complained in apparent surprise:

> Lotus wants to again extend it [the law]. By now, the explanation that Lotus and other plaintiffs in copyright suits are trying to define the limits sounds a bit hollow. The limit was established with *Paperback*. Why take it further with Borland?
>
> Where will the litigation end? If the *Paperback* ruling's limits were not enough, what will be? And what about Apple or Ashton-Tate? Lotus is not alone in court today. Virtually anyone with a product that has earned some recognition—enough to be used as a design target—is considering legal moves.
>
> ... it seems an awfully rough road to travel in the pursuit of some guidance.

The second concern that exists in the software community is based on doubt that courts can over the long run sensibly apply copyright law to user interfaces and other noncode aspects of software. The *Paperback* decision is a setback, but it cannot be called the last straw. A few more decisions like it, however, may seriously retard software progress by misguided and excessive expansion of copyright monopolies until the courts broadly preempt competitive use of functional advances in the field.

Defenders of this trend say, "Don't fix it if it ain't broke." But if we have a few more *Paperbacks*, somebody will have to declare the system broken. (Some observers contend that those who say "Don't fix it if it ain't broke" cannot recognize an already-broken system when they see it.) At that point, copyright theory will have been shown to conflict with the lessons that the school of hard knocks has taught us. Copyright doctrine holds that copyright protects only "expressive" aspects of works, not ideas or systems embodied in works. The same doctrine says that copyright does not give control to copyright owners over functional aspects of works. But rulings like *Paperback* show us that copyright law cannot effectively promote and encourage technological advance without betraying the promises of copyright doctrine.

References

1. 740 F.Supp. 37, 15 U.S.P.Q2d 1577 (D. Mass. 1990).
2. R.H. Stern, *IEEE Micro*, June 1989-Feb. 1990, various pages.
3. Rachel Parker, "State of the Industry: Lotus's Copyright Protection Is Turning Into a Feeding Frenzy," *Info World*, July 9, 1990, p. 42.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 177 Medium 178 High 179

Call for Papers

IEEE Micro seeks general interest manuscripts for 1992 issues.

Suggested topics include biological computing, VHDL design and workstations, operating systems, multiprocessing, optical computing, microcomputing to aid the handicapped, and HDTV. Submit six copies of manuscripts to either Editor-in-Chief Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129 Torino, Italy, or to Associate Editor-in-Chief Ashis Khan, Mips Computer Systems, Inc., 950 DeGuigne Drive, Sunnyvale, CA 94086; Internet: ashis@mips.com.

Micro Standards

Sorry, Captain Kirk!

Carl Warren

McDonnell Douglas

Space Systems

(714) 896-3311

x. 6-0669

MCI Mail: 310-9380

In the spirit of April's Fools Day, I take a more light-hearted approach to standards than usual in discussing a recent conversation with one of my coworkers, Joe Smith.

Joe dropped over and sat down in my 4 × 8-ft domain to discuss the problem of rating the good qualities of a machine. Like most of the engineers here at McDonnell Douglas, Joe has his own special interests. He prefers the development of communications standards for space-to-space and space-to-ground transmissions. In one of his working groups the issue of establishing a merit system of computers popped up.

"What are the various ways you rate a computer, Carl?" he asked. I responded that over the years various methods have been developed. The most typical method performs some form of a benchmark. "The problem is," I continued, "that benchmarks are like Tribbles," carefully putting it in terms close to Joe's unique sense of humor and referring to a now-famous Star Trek episode in which the ubiquitous Tribbles first appeared.

I went on to note that like the Tribbles, benchmarks are everywhere. And, of course, Benchmark A is far more important and useful than Benchmark B—yet like a Tribble, similar and familiar.

What transpires is an *Entscheidung* problem—an A,B type of decision. Here, when A is right, B is right as well, and both are wrong at the same time—obviously a terrible paradox. For example, consider a system composed of some backplane capable of managing *n* levels of transfers per second, a speedy processor, and some I/O control. Your goal is to determine how well the machine performs mathematical operations. In reality, you want to know how well the processor will exchange information from register to register, register to memory, and back again.

You can take a number of approaches. You might perform a Sieve sort, or perform a Dhrystone test (see *IEEE Micro*, Oct. 1988, p. 64-75). The result will be some number that is a figure of merit.

But that figure needs some form of interpretation to make sense. So, you decide, based on some incantations known only to you, that high is good and low is bad, or vice versa. In this case it really makes no difference which you decide upon, since the only point I'm trying to make is that the number is determined by your point of reference, thus the *Entscheidung* problem. Is decision A better than B? Are they equal? Or, do they both indicate failure thus negating their own purpose?

Of course, by now Joe was utterly fascinated, and I was warming to the subject. So I continued.

"Once you have decided that you want to measure a system, you have to decide what to measure, when, how, and what the dickens the measurement means. The determination is a factor of how you define success or failure."

"This all seems very confusing," Joe interjected. "It appears you are saying that trying to measure a system performance produces the ultimate paradox."

I responded that he was very astute and that was precisely what I was getting at. A benchmark when attempted by the uncultured does indeed produce the ultimate paradox. Good becomes bad and bad becomes good. "But, there is a way out of the woods," I exclaimed, with no small degree of exuberance. "We call this path standards," I told him.

The significant elements of a standard remove doubt, eliminate semantic and mathematical

0272-1732/91/0400-0034$01.00 ©1991 IEEE

paradoxes, and establish some order to the world. It is probably safe to assume that even the caveman had some reverence for standards.

"Joe, imagine yourself as Glug, living in the marshes of a fresh new world. You have, in some way, discovered fire. And being an above-average caveman, you have found a way to start fires using stones that have a special look, along with mice nests of a dry material. Every time you tried something else, it failed. Since eating a cold microwave dinner wasn't on the agenda, you quickly settled on one standard methodology for starting your fire. When you passed this on to a traveling tribe, they discovered that opening up a facet on one of the rocks made it easier to get a spark. Aha! You have a revision to the base standard—a better definition."

By now, I knew I had Joe. He was so interested that he even offered to buy coffee.

The issue of measuring system performance is indeed interesting. And the methodologies for doing so are wide-ranging. Unfortunately, so are the rules. What is apparently valid for one company may not fill the bill for another—simply because the benchmark shows their system in a bad light.

Some interesting elements that have to be taken into account when measuring a system include:

1) the characteristics of the bus architecture,
2) the characteristics of the memory system,
3) the characteristics of the processor and processor instruction set,
4) whether the system is to be measured with the optimum Gibson mix (instructions combination), and
5) the characteristics of the I/O system.

Additionally, you have to come to some consensus about what you are measuring. If mathematical manipulation is what you are interested in, I/O plays no role. Registers and memory must be considered, as well as clocking, software latencies, and the Gibson mix of the instructions being used to perform the calculations.

As an experiment, I asked Joe to develop a method of measuring the following general-purpose system for various functions he might be investigating: A 20-MHz 386 microprocessor on a backplane bus capable of sustaining a 40-Mbyte/s burst transfer rate, 20 sustained. Accompanying it is a memory system with an 80-ns DRAM, an 11-ns static RAM for cache, and an I/O system of 156 Kbytes/s.

This question may interest you as well. Drop me a note on the approach you would take at McDonnell Douglas, 5301 Bolsa Ave., MS A95-J845-17/6, Huntington Beach, CA 92647, or via my e-mail address. I'm looking for the way you would define the standards for resolving the problem.

Yes, you can choose an existing benchmark such as Whetstone, Dhrystone, Linpack, and so on. However, seriously consider what the results actually mean.

Remember, avoid the Tribble pitfall.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 180 Medium 181 High 182

New Products

Send announcements of new microcomputer and microprocessor products to Managing Editor, IEEE Micro, PO Box 3014, Los Alamitos, CA 90720-1264.

Joe Hootman

University of North Dakota

Miscellanea

Debugging connections

The LA-Connect debugging system combines a logic analyzer from Tektronix with Microtec Research's source-level debugger and C, C++, and Pascal cross-compilers. Used with complex- and reduced instruction set computing-based embedded systems, the product features code generation and debugging solutions for the Motorola 68xxx family and the Intel 960. LA-Connect performs debugging functions via a RS-232 connection between a workstation and the target system. *Tektronix; from $21,000 each.*

Reader Service No. 10

Glare spray

A spray-on application for computer monitors and TV sets provides another choice in reducing screen glare, reflection, and ultraviolet emissions. Glare Shield comes in a three-ounce aerosol can; when applied, the coating is permanent. *Computer Safety Products.*

Reader Service No. 11

Electronic components catalog

The *High Performance Electronics Selection Guide* disk includes listings of more than 1,250 current component models, technical literature, domestic prices, and an industry cross-reference section. Users can search and select part numbers in 11 categories. They can view a separate new product section and download product updates from an electronic bulletin service. *Burr-Brown Corp.; free.*

Reader Service No. 12

Compact hard disks work with laptops

Measuring 1.25 × 3 × 5.5 inches, the Pocket Disk PD20-1 is a removable 20-Mbyte hard disk drive for IBM PC-compatible computers. According to the company, the hard disk averages access times of 23 ms. The PD20-1 and the slightly larger size PDH20-1 connect externally to desktop computers. The PDH20-1 works with selected laptops. *Tradewinds Peripherals; $895 each (PD20-1), $950 each (PDH20-1).*

Reader Service No. 13

Two-way conversions

The Model 109 bidirectional converter can connect a computer with parallel output to a serial device or vice versa. The unit contains 64 Kbytes of data memory, which releases the sending device for other uses during printing. Model 109 accommodates serial data rates from 300 to 38,400 bps. Dual-in-line package switches set the operational parameters, which include conversion direction, baud rate, parity, character length, and handshake mode. *Telebyte Technology; $99 each; available from stock.*

Reader Service No. 14

Telebyte Technology's Model 109 bidirectional converter

Parallel-port hard drives

Four external hard drive subsystems attach directly to the parallel port of PC-compatible computers. All drives include pass-through capabilities to allow a printer to function simultaneously with the drive. Two of the Parallel Line drives measure approximately 5 × 4.5 × 1.5

inches; the other two are about 7.5 × 4 × 1.5 inches. Models range from 20 to 100 Mbytes and battery-powered units are available. *Pacific Rim Systems; from $799 each.*

Reader Service No. 15

AC power supply

The Universal Input Tabletop Switching Power Supply XT 25 operates on an AC power source from 90 to 250 volts of alternating circuit without jumpers or switches. Available in single-, dual-, and triple-out models, the XT 25 comes in a plastic housing for external use with the equipment being powered. *Converter Concepts.*

Reader Service No. 16

Software

C and C++ creations

Using a set of cross-compilers for the Motorola 680xx microprocessor family, developers can develop real-time applications in the C or C++ languages. Compatible with AT&T Release 2.0, the VRTX compiler uses type checking and object-oriented programming concepts. Other features include a smart loader for dynamic address allocation in RAM or ROM memory, automatic access to programs in ROM, and testing capabilities. *Microdata Soft.*

Reader Service No. 17

Software source book

The *Motorola 88000 Directory* catalogs more than 1,200 software applications for the 88000 family of RISC microprocessors. Entries include more than 300 88/Open consortium-certified applications and approximately 1,500 88000-based software programs. The directory also contains lists of hardware systems, board manufacturers, operating systems, compilers, and 88000 services. *Motorola.*

Reader Service No. 18

Kernel for Intel processors

The C Executive real-time operating system kernel supports Intel's 960 CA and KB processors and allows designers to execute multitasking applications written in C. Standard I/O device drivers and a C library come in the package. Other options include a file system and programs for DOS 8086 file replication and system debugging. *JMI Software Consultants.*

Reader Service No. 19

Ada development software

Overcoming 640-Kbyte DOS and 286 limitations, Alsys' Ada Software Development Environment for the 486 supports 32-bit-mode code and data addressing. The program consists of a compiler with optimizers, binder, multilibrary environment with family, library, unit managers, and a tool set. Any 486 machine running a DOS version 3.0 or higher can serve as the environment's host. *Alsys; $4,995 each.*

Reader Service No. 20

Neural-network spreadsheet

Braincel, a neural-network program for PCs embedded in a Microsoft Excel spreadsheet, allows users to preprocess and test data. The program uses both back-percolation and back-propagation algorithms to train neural networks and open up real-time, neural-net modeling of nonstationary, nonlinear systems.

Twenty new Excel function macros come with Braincel, which can import data from Windows 3.0 applications. Braincel occupies 320 Kbytes of RAM memory and requires Windows, DOS, and Excel; a mouse is recommended. *Promised Land Technologies; $249 each, $99 each to Windows Users Group Network members.*

Reader Service No. 21

Fortran interface

App Maker users can create a Macintosh interface for Fortran applications by clicking and dragging onscreen items under Fortran Tools for App Maker. The program permits App Maker to generate Fortran source code for compiler use in selected modules or an entire application.

Programmers can create customized windows, menus, dialogs, or alerts and integrate them in existing source code. *Language Systems; $100 each (App Maker 1.1 owners), $295 each (bundled with Fortran tools).*

Reader Service No. 22

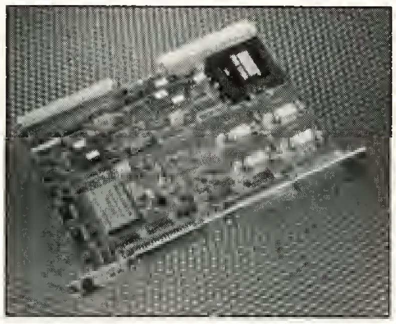
Boards and special components

A/D conversions at 40 KHz

The XVME-545 Analog I/O Module performs 16-bit A/D conversions at 40 KHz. Designed for high-resolution and control applications, the module includes 32 single-ended, 16 differential, or 32 psuedo-differential input channels, and four analog output channels.

The XVME-545 can operate in five modes: random, single-channel, sequential, external trigger, or loop-back testing. Users can configure the four 12-bit analog output channels for voltage ranges of 0-5V, ±2.5V, ±5V, and ±10V. *Xycom; $1,700.*

Reader Service No. 23



Xycom's XVME-545 Analog I/O Module

Cache access

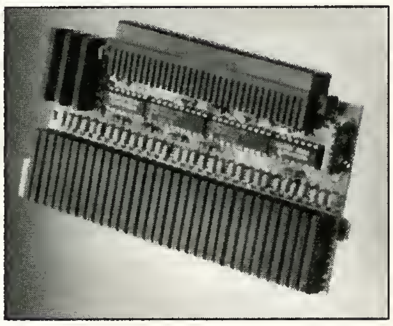
A 5.25-inch form-factor peripheral device operates as a front end to large-capacity SCSI disk drives. The SA807 Cachebox accesses data by caching the most frequently requested data.

The device offers 8 to 64 Mbytes of cache memory in increments of eight or 16 Mbytes per board. It installs between the system's existing SCSI port and the disk drive. *Dilog; from $5,600*

each (18-Mbyte versions; 30 days ARO.
Reader Service No. 24

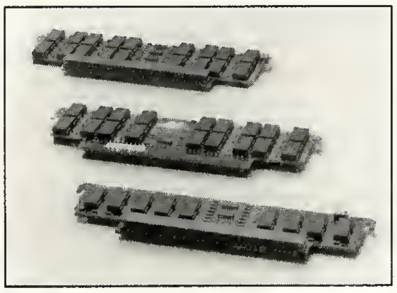
HP memory expansion

A 4-Mbyte expansion board for Hewlett-Packard 340 computers provides additional memory for Unix, Pascal, and Basic applications. According to the company, the Micro RAM 340 board installs into an expansion slot in 15 minutes. Each board comes with documentation and a two-year warranty. *Intelligent Interfaces; $995.*
Reader Service No. 25


Intelligent Interfaces' expansion board

Workstation memories

A line of error-checking and error-correcting modules expands HP/Apollo workstation memories from 64 to 128 Mbytes, depending on the system. The modules, which come in 4-, 8-, 16- and 32-Mbyte increments, fit into HP 9000 Series 345 and 375 workstations, and HP/Apollo 400, 425, and 433 workstations. *Martech; approximately $175 per Mbyte (32-Mbyte configuration).*
Reader Service No. 26


Martech's memory expansion boards

Futurebus+ CPU module

The NR3000-1 CPU module offers a 25-MHz, Mips R3000 microprocessor for use with Futurebus+ products. It supports both cache-coherent-memory and message-passing multiprocessor systems. The NR3000-1 includes a two-level cache hierarchy of 128 Kbytes of primary cache, 1 Mbyte of secondary cache, and 1 Mbyte of both local RAM and erasable, programmable, read-only memory. The package also contains two serial ports, timers, a real-time clock, and a local bus expansion slot. *Nanotek; $9,900 (100s).*
Reader Service No. 27

Network software

Network monitoring

Two network products provide management and control for communication networks. The NMC-90 controller, operating on a Unix-based graphics workstation, provides X11 windowing, configuration, fault, account, performance, and security management capabilities for intelligent, diagnostic data set networks. The Megaview Version 2.0 software package features alarm filtering, multiple printer support, severity assignment to alarm events, and database storage. *General Datacomm; $43,550 (NMC-90), $30,500 (Megaview).*
Reader Service No. 28

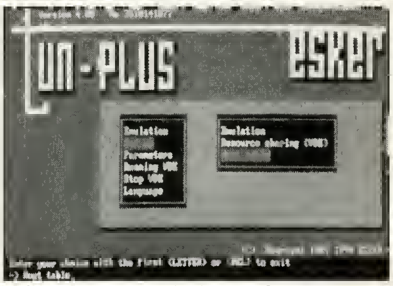
Unix System 5 interface

The Netbios interface allows MS-DOS LAN applications to migrate to the Unix System 5. Netbios' sources include more than 20,000 lines of code and 250 routines. Netbios requires 40 Kbytes of memory when installed. *The NTI Group; pricing not available.*
Reader Service No. 29

Emulators

Terminal conversions

The Tun (Terminal Universal) communications software program converts MS-DOS-based PCs into multistation terminals via configured emulations. Available in English, French, or German versions, Tun features VT52, VT100, VT220, ANSI, 386 AT, and other standard emulations. The software allows for the management of Com 1 and 2 serial ports on one connection. It also controls a modem card for remote connections, transfers files via a protocol, and permits users to switch from one host computer to another. *Esker.*
Reader Service No. 30



Esker's Tun emulation software

VT420 emulator

The KEA Term 420 software program emulates the VT420 terminal, supporting it with double high and wide characters, 132-column mode, multiple-page features, and coupling capabilities. It runs under Windows 3.0 and includes extensions that go beyond the VT420: ISO colors, attribute mapping to colors, VT340 block mode and local editing, and variable screen length from 24 to 144 lines. While permitting access to VAX and Unix text-based applications, Kea Term 420 allows downward compatibility with earlier VT terminals. *KEA Systems; $195 (until July 1, 1991).*
Reader Service No. 31

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 189 Medium 190 High 191

Product Summary

Joe Hootman
University of North Dakota

Manufacturer	Model	Comments	R.S.#
New devices			
Integrated Device Technology	Dual-port RAMs	Four devices offer 30-ns access times and feature dual-port memory cells and on-board arbitration logic. Suitable for high-speed multiprocessor applications, the RAMs come in configurations of 8K × 8 (IDT7005), 16K × 8 (IDT7006), 4K × 16 (IDT7024), and 8K × 16 (IDT7025). From $93.71 (100s).	80
Texas Instruments	ACT2235 and ACT2236 FIFOs	Two bidirectional, asynchronous FIFOs feature a high-output drive for direct bus interface. They are capable of operating at frequences up to 50 MHz with a 25-ns access time. The 1K × 9 × 2 FIFOs are cascadable in width and available in 44-pin PLCCs. From $22.32 (1,000s).	81
Philips Components	PCB80C51BH-5-30 microcontroller	According to the company, its 30-MHz version of the 80C51 executes almost 60 percent of instructions in 0.4 μsec and the remaining percentage in 0.8 μsec. The microcontroller consumes 44 mA at 5.5V supply voltage. The 5-30 comes in a 40-pin dual-in-line, 44-lead PLCC, or 44-lead quad flat-pack package. Samples available; Hfl5.40, volume quantities (10,000s), 10 weeks ARO.	82
Mitsubishi Electronics America	DRAM chips	Inside thin, small-outline packages, the 4-Mbyte DRAMs offer 60-ns access times and 200-μA maximum standby currents. They come in 4 Mbytes × 1 or 1 Mbyte × 4 organizations and in the following modes: fast-page, nibble, static column, write-per-bit and fast-page, and write-per-bit and static column. $34 (100s).	83
Burr-Brown Corp.	D/A converters	The DAC813 12-bit, microprocessor-compatible converter dissipates a maximum of 270 mW. It includes a precision 10V reference, voltage output amplifier, and microcomputer interface logic. The dual 16-bit DAC725 converter incorporates double-buffered data latches, internal buried-Zener references, and low-noise output operational amplifiers in a hermetic 28-pin package. From $11.90 (100s, OEM); available from stock.	84
Motorola	MC145170 PLL synthesizer	With serial interface capability, the 16-pin frequency synthesizer directly interfaces to voltage-controlled oscillators in the MF, HF, and VHF bands. It includes Bit Grabber registers, tuning via a 2-byte serial transfer to the 16-bit N register, and programmable R and N counters. $4.17 (500 to 999).	85
Philips Components	VTCXO oscillator	The 4-mm VTCXO is a temperature-compensated crystal oscillator with a volume of 0.85 cm^3. The oscillator generates frequencies between 8 and 16 MHz and consumes a low maximum current of 3 mA. Hfl40 (5,000s), six weeks ARO.	86

Cache memory design

continued from p. 19

and clean, it becomes private and clean; when it is shared and dirty, it becomes private and dirty.

Read or write miss. When a cache miss occurs upon a read or a write operation, the cache first replaces a cached copy and then executes the requested operation, such as a hit condition. The replacement phase consists of the

- selection of a victim cache block;
- updating of the memory block pertinent to the victim copy (that is, the copy stored in the victim cache block) when it is necessary; and
- reading of the requested memory block by means of a read block transaction.

When the victim copy is dirty (private or shared), the cache writes it into the shared memory by an update block transaction. Finally, the cache executes the requested operation. Specifically, when we have a miss on a read operation and the memory block is not shared (the shared line is not active during the read block transaction), the copy state becomes private and clean. Otherwise, the cache holds the copy in the

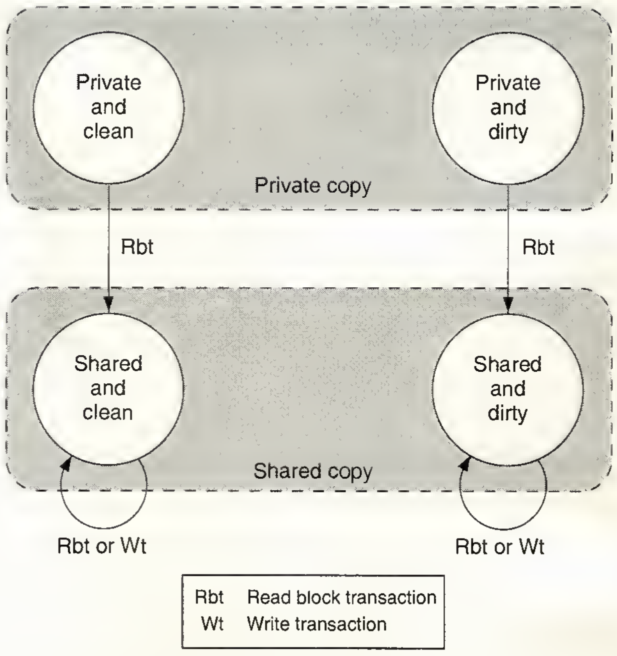


Figure 3. State transitions of a cached copy involved in a bus transaction.

Table 1. Transactions needed to execute a processor operation.

Condition	Read operation Number	Read operation Transaction type	Write operation Number	Write operation Transaction type
Hit, requested copy is private	0	—	0	—
Hit, requested copy is shared	0	—	1	Write
Miss, victim copy is clean, requested copy is private	1	Read block	1	Read block
Miss, victim copy is clean, requested copy is shared	1	Read block	2	Read block, write
Miss, victim copy is dirty, requested copy is private	2	Update block, read block	2	Update block read block,
Miss, victim copy is dirty requested copy is shared	2	Update block, read block	3	Update block, read block, write

Write transaction: The cache broadcasts a write operation
Read block transaction: The cache acquires the copy of a memory block
Update block transaction: The cache updates a memory block

shared and clean state. When a miss occurs during a write operation, the cache holds the copy either in the private and dirty state (if the shared line is not active) or in the shared and clean state (if the shared line is active). The cache updates the copy in all cases and, if the memory block is shared, the write operation is also broadcasted on the common bus.

Table 1 summarizes the number and the types of transactions needed to execute a processor operation.

Common bus transactions. All of the caches disregard the update block transactions. On the contrary, during a read block or a write transaction, each listening cache verifies whether it holds the copy of the memory block involved. In the affirmative case, it activates the shared line and, possibly, modifies the value and/or the state of its copy (see Figure 3).

During a write transaction the cache updates only its copy of the memory block involved. During a read block transaction, the cache changes the copy state into the shared and clean or shared and dirty state, depending on whether the copy is private and clean or private and dirty. When the copy is dirty, the cache disables the shared memory and supplants it by supplying data during the transfer phases of the bus transaction.

Protocol comparison

Many proposals describe cache memories for multiprocessor systems. The characteristics and operational features of these protocols depend on the multiprocessor architecture for which they are proposed. Goodman[26] provided the first solution to the coherence problem; Archibald and Baer[9] and Sweazey and Smith[14] surveyed other solutions.

Archibald and Baer examined the efficiency of several solutions by using a simulation program. They demonstrated that the Illinois,[13] Dragon,[18] and Firefly[27] protocols give better performances than the Berkeley,[12] write-once,[26] and Synapse[28] protocols because the first three protocols dynamically discriminate between private and shared copies. Furthermore, the three latter protocol solutions sometimes require either a bus transaction for a write operation on a private copy (a burden for the common bus and more time consuming) or the invalidation of the other copies before a write transaction. (Invalidation of actual shared copies causes misses on the succeeding references to such copies in caches in which the copy invalidation occurs.)

Archibald and Baer also demonstrated that the distributed write approach of Firefly and Dragon offers the best performance for handling shared copies.

Firefly protocol. Digital Equipment Corp. developed the Firefly protocol for the Firefly system, which is a multiprocessor workstation.[9, 27] In this protocol (see Figure 4), a valid copy resides in one of three possible states: valid-exclusive (unmodified and unique), shared (unmodified and other copies can exist in the caches), or dirty (modified and unique).

Dirty copies are the only ones written back to shared memory upon replacement. For a write operation on a dirty or a valid-exclusive copy, the cache updates only the copy. In addition, the status of a valid-exclusive copy changes to dirty. If a copy is shared, the cache also broadcasts the write operation on the common bus to update the memory block and the other copies.

For a miss during a read operation, the cache acquires a copy by means of a read block transaction. During this transaction, all the listening caches with a copy of the requested memory block supply it to the requesting cache and activate the shared line. If a listening cache holds the copy in the dirty state, this cache supplies the copy to the requesting cache and updates the main memory at the same

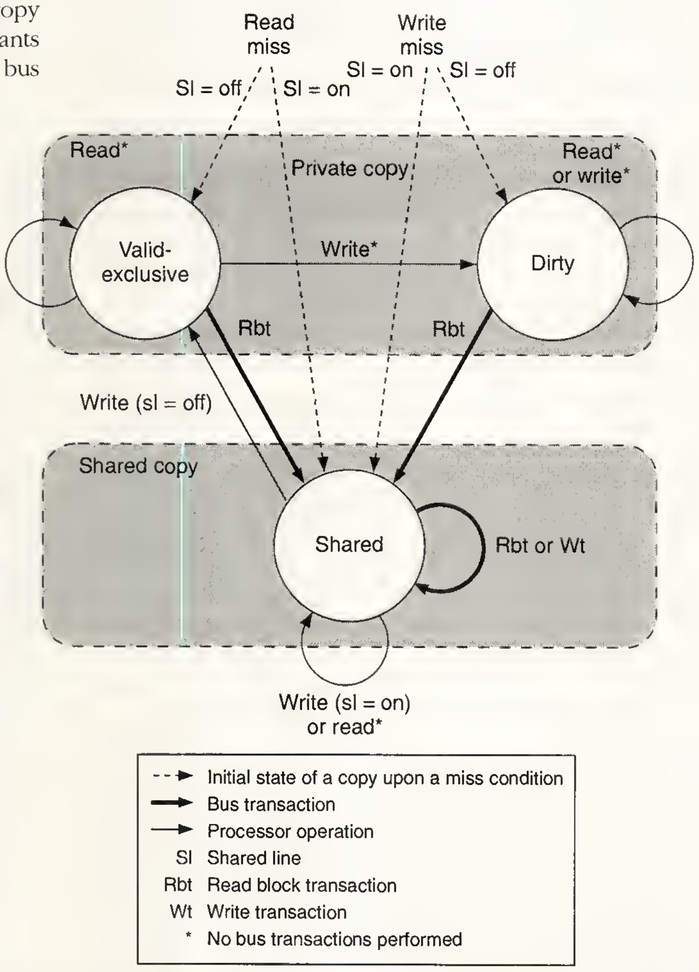


Figure 4. State transitions for the Firefly protocol.

time. If the shared line is active, all caches (including the requesting cache) set the copy state to shared. If the shared line is not active, the requesting cache sets the state to valid-exclusive and the main memory supplies the copy of the memory block. In the same way, for a write miss, one or more caches or the main memory supply the memory block. If the shared line is not active, the copy assumes the dirty state; otherwise, the copy assumes the shared state and the requesting cache broadcasts the write operation on the common bus.

Dragon protocol. The Dragon protocol takes its name from the homonymous multiprocessor designed at the Xerox Palo Alto Research Center.[9, 18] This protocol (see Figure 5) has four possible states for a valid copy: valid-exclusive (unmodified and unique), dirty (modified and unique), shared-clean (unmodified and shared), and shared-dirty (modified and shared).

Like Firefly, this protocol allows multiple processors to write on shared copies, but the shared memory is disabled during the write transaction. This strategy requires the addition of the shared-dirty state, which does not have a corresponding state in the Firefly protocol. The cache with a copy in the shared-dirty state must update the relevant memory block before the destruction of the copy. Also, the cache must substitute the shared memory upon the occurrence of a read block transaction involving the memory block pertinent to a shared-dirty copy. As in Firefly, each cache activates the shared line when it has a copy of the memory block involved in the bus transaction.

The cache locally performs a write operation on a dirty or a valid-exclusive copy. Also, when the copy is in the valid-exclusive state, it changes to the dirty state. Otherwise, when the copy is in one of the shared states, the cache broadcasts the write transaction to update the rest of the copies. During this write transaction, if the copy is shared-clean and the shared line is active, the following state transitions occur: from shared-clean to shared-dirty in the cache performing the write operation, and from shared-dirty to shared-clean in the possible listening cache with the copy in the shared-dirty state. If the shared line is not active during the write transaction, the local copy assumes the dirty state.

For a read miss, the cache acquires a copy by means of a read block transaction. During this bus transaction, if a listening cache has the copy in the dirty or shared-dirty state, such cache asserts the shared line, supplies the data, and, if the copy state is dirty, changes it to shared-dirty. On the other hand, if a cache holds the copy in the valid-exclusive or shared-clean state, it asserts the shared line and the state of a valid-exclusive copy changes to shared-clean. In both of these latter cases, the requesting cache stores the copy in the shared-clean or valid-exclusive state in conformity with the state of the shared line.

For a write miss, the requesting cache stores the copy in the dirty or shared-dirty state to conform to the state of the shared line. When the copy is stored in the shared-dirty state, the cache performs a write transaction.

Differences between RST, Dragon, and Firefly. We derived the RST coherence protocol from the Dragon after modifying it to simplify its implementation and improve its performance.[29] In our protocol, unlike Dragon, every write transaction updates the memory block involved. In this way a memory block on which several processors execute write operations may already be updated. This strategy allows us to minimize the number of update block transactions needed for every processor and to simplify the protocol and its implementation.

For example, processor P_i reads a memory location belonging to memory block m. If cache C_i (based on the RST protocol) does not have the copy of m, it acquires a copy through a read block transaction. If at least one other cache has the copy of m, the local copy will be shared (shared line = on), otherwise it will be private (shared line = off).

Let us consider the latter condition: C_i stores the copy in the private and clean state. When another processor P_j executes an operation on a memory location belonging to m, cache C_j acquires the copy of m and both copies in C_i and C_j assume the shared and clean state. Afterwards, any write operation executed by P_i or P_j needs a write transaction that updates all the cached copies and the memory block m. In these conditions, the replacement of the copy of m in C_i or C_j does not require an update block transaction.

On the other hand, the same sequence of operations on the Dragon produces a shared-dirty copy in the cache relative to the processor that executed the last write operation. Therefore, the replacement of this shared-dirty copy needs an update block transaction.

Finally, for a write operation on a shared and clean copy, when the shared line is off during the write transaction, RST produces a private and clean copy. Dragon produces a dirty copy. RST minimizes the number of dirty copies and, consequently, the number of update block transactions. RST is also simpler than Dragon since it does not have shared-clean/shared-dirty transitions (see Figures 2, 3, and 5).

Like RST, Firefly's strategy involves updating the memory block during the write transaction. However, Firefly has two drawbacks:

- The time taken by a write transaction is greater because the access time of the shared memory is longer than that of a cache memory working in the slave mode.[9]
- During a read block transaction, all caches with a copy must respond by putting the data on the bus. This solution exaggerates the problem of the glitches on the handshake signals. These glitches generally occur when several drivers[14] simultaneously operate on the handshake signals.

In the RST protocol, we overcome these drawbacks as follows:

- The shared memory includes a data buffer. During a write transaction, the shared memory stores the data in the data buffer and, afterwards, updates the memory block. Therefore, the access time of the shared memory during a write transaction is equal to a cache memory working in the slave mode.
- During every bus transaction, only one unit operates as a slave on the common bus. In particular, for a read block transaction, this unit is the cache involving a dirty copy of the block. In all other cases, it is the shared memory. This strategy makes it possible to simplify the interface modules to the common bus and minimizes the glitches on the handshake signals.

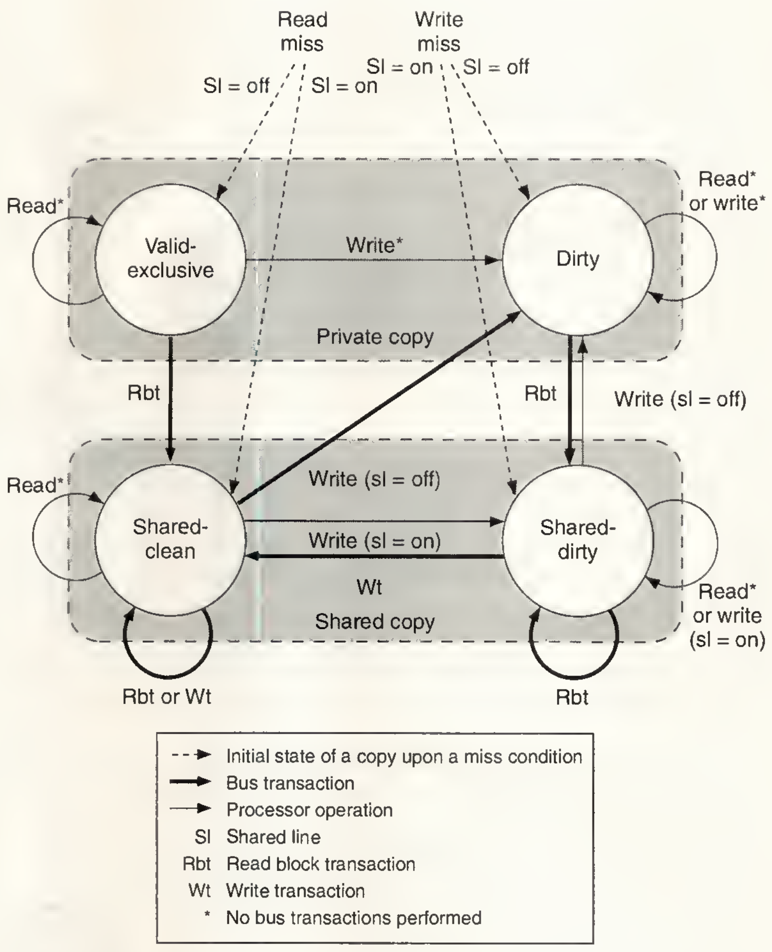


Figure 5. State transitions for the Dragon protocol.

The first solution does work in Firefly, though the second is not applicable because Firefly does not distinguish between clean and dirty copies of the shared copies. In practice, during a read block transaction, Firefly considers all shared copies as dirty.

Simulation program. During the RST design we used a simulation program to adjust the cache memory for maximum system performance and a minimum of bus requirements for each cache. The simulation model we adopted is an extension of the model proposed in Dubois and Briggs[11] and used by Archibald and Baer.[9] It is also capable of simulating kernel activity (for a Unix-like[30-32] operating system[33]), context switches, and process migration.

We base the simulation program on a synthetic reference stream, which is a sequence of references obtained from a stochastic model of program execution and from a multiprocessor model. Program, system, and processor parameters describe both models.

(Simulations based on actual traces are not applicable to multiprocessor performance evaluation, since traces for multiprocessors are either not available or bound to a specific multiprocessor architecture.[9] Furthermore, a concurrent program generates different traces when run on different multiprocessor architectures because of the different timing relationship between the concurrent processes.)

The program parameters characterize the application software. For each program they are the:

- number of processes;
- size of the user code, private data, and shared data of every process;
- frequency of references to each of these regions; and
- referencing behavior in each region.

Referencing behavior is the probability $P_{d,r}$ that, when a location l in a region r is referenced at time i, location $l + d$ in the same region is referenced at time $i + 1$. We assume that the probability does not depend on either l or i. In code areas, $l + d$ (where d is positive) has the maximum probability of being the next referenced address, and the probability is approximately normal with mean 0 in data areas.

The parameters of the system characterize the operating behavior of the system. The stochastic model uses these parameters to obtain a specific reference stream. They are the:

- size of the kernel code, kernel, and communication data;
- frequency of references to each region;
- referencing behavior in each region;
- frequency of context switches;
- probability that a kernel routine is running; and
- time slice.

The parameters of the processor characterize the CPU instruction set. They consist of the percentage of read operations, average number of clock cycles per instruction, and probability distribution of the number of memory operations per instruction. Before each simulation, the program asks for the number of instructions that the multiprocessor must execute and certain parameters that characterize the cache and multiprocessor architecture. These parameters include the timing of each bus transaction and the cache's inner action, coherence protocol (Dragon, Firefly, or RST), block size, cache capacity, degree of set associativity, CPU clock frequency, and number of processors.

The multiprocessor model is identical for each protocol. In particular, each cache memory includes a write operation buffer and the shared memory contains a data buffer. When the processor requires a write operation, the cache stores the operation in its buffer and signals the completion of the operation. Afterwards, the processor executes the operation held in its buffer. The data buffer speeds up the shared memory and, in particular, minimizes the time needed for both write and update block transactions.

The working of the simulator. At the beginning, the simulator allocates the memory segments for executing every process in the common memory space, creates the virtual-to-physical mapping table, and starts the simulated executions of the programs. A queue holds the processes that are ready. On the basis of a first-in, first-out policy, the scheduler extracts a process from the queue to run when a processor becomes available. A context switch occurs when the time slice allotted to the process expires.

To simulate the execution of an instruction, the simulator creates a stack of length L (where L is the average number of clock cycles per instruction, excluding wait states). Each stack entry specifies whether the issuing of a memory reference will occur in the clock cycle corresponding to the entry.

The number and distribution of memory references for each instruction are generated stochastically by means of the parameters of the processor. The simulator reads a stack entry at every clock cycle. When the entry is a memory reference, the simulator generates an address in three steps:

- selection of the region;
- generation of the virtual address within the region, depending on the region and the process issuing the reference; and
- generation of the physical address.

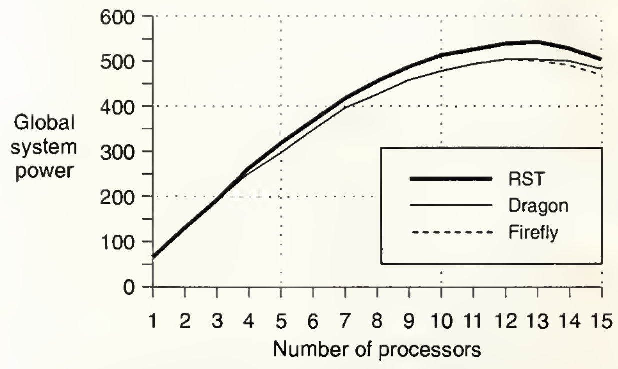


Figure 6. Global system power versus the number of processors in the case of 1 percent of references to shared data.

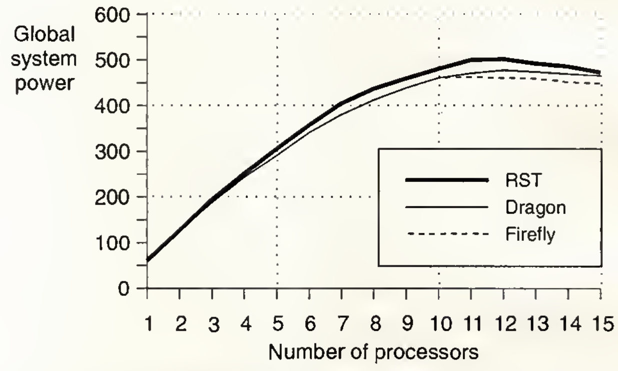


Figure 7. Global system power versus the number of processors in the case of 10 percent of references to shared data.

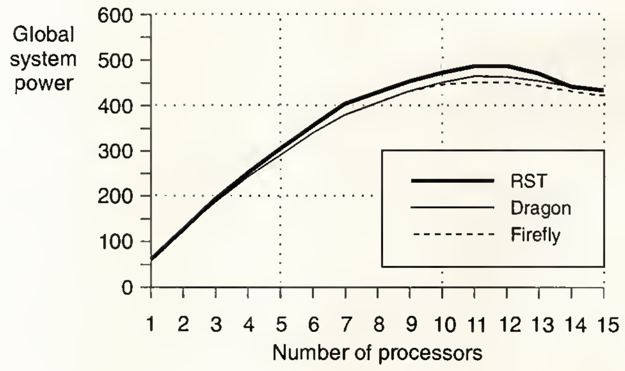


Figure 8. Global system power versus the number of processors in the case of 25 percent of references to shared data.

Table 2. Main input parameters.

Parameters	Features	Number
Program	Number of processes	20
	Size of user code of each process	64 Kbytes
	Size of private data of each process	16 Kbytes
	Size of shared data of each process	4 Kbytes
	Frequency of references to shared data	0.01, 0.1, 0.25
Kernel	Size of the kernel code	128 Kbytes
	Size of kernel and communication data	64 Kbytes
	Probability that a kernel is running	0.1
	Frequency of process context switches	10,000 per second
Processor*	Percentage of read operations	0.75
	Average number of clock cycles	14
	Probability of zero memory reference	0.1
	Probability of one memory reference	0.3
	Probability of two memory references	0.6
	Clock frequency	10 MHz
Cache	Cache capacity	64 Kbytes
	Block size	16 bytes
	Degree of set associativity	Two-way
Common bus*	Duration of the read block	1,200 ns
	Duration of the update block	1,200 ns
	Duration of the write transaction	300 ns

* Processor and common bus parameters are obtained from standard microprocessor and multiprocessor systems.

We choose between the regions according to the respective probabilities. Then a function, whose probability distribution depends on the region chosen, generates the virtual address. Finally, the simulator uses the virtual-to-physical mapping table to generate the physical address. The processor puts the memory request in the service queue of its cache.

Each cache services memory requests by determining whether the requested block is present or absent. When the copy is present, after one clock cycle the cache sends the processor a command to continue. When the copy is absent, the cache performs all the actions needed to acquire a copy of the requested memory block and sends the processor a command to continue. The cache also executes actions relevant to a bus transaction as specified by the selected coherence protocol. Actions relevant to a processor operation and a bus transaction are executed in parallel except when both need to operate in the data fields or need to modify the block fields.

At the end, the simulator produces:

- statistical data for each processor, including the actual average number of clock cycles per instruction; hit ratio; number of replacements; and number of write, read block and update block transactions. It also includes actual sharing (ratio of the memory operations on shared copies to the total number of memory operations) and word transfer ratio (the ratio of bus word transfers generated by the cache on the common bus to the word transfers needed if there were no cache); and
- global statistical data, such as the global system power (the sum of processor utilization measured by the ratio of time spent accomplishing useful work to the total execution time), bus utilization ratio, and total number of context switches.

Simulation results. Figures 6, 7, and 8 show the global system power (multiplied per 100) for all protocols with one to 15 processors, where the percentage of references on shared data is 1, 10, and 25. (Table 2 summarizes the values of the main input parameters.) The average improvement of the global system power of RST to both the Dragon and the Firefly protocols is greater than 4 percent.

Analysis of all the simulation outputs justifies the RST's improved performance in comparison with Dragon by the

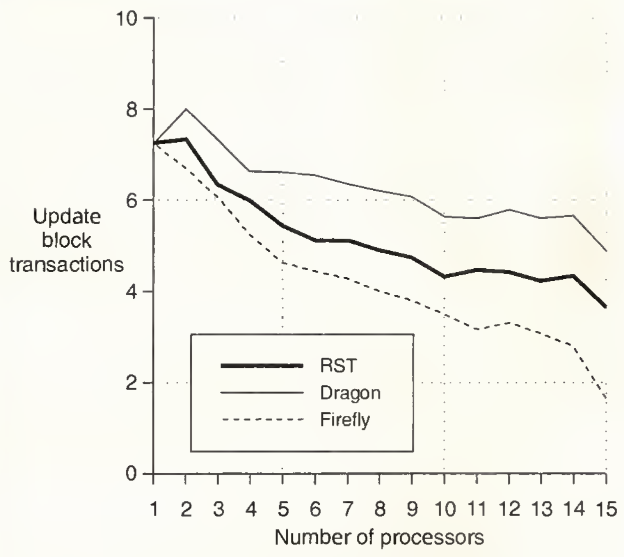


Figure 9. Update block transactions for 1,000 operations.

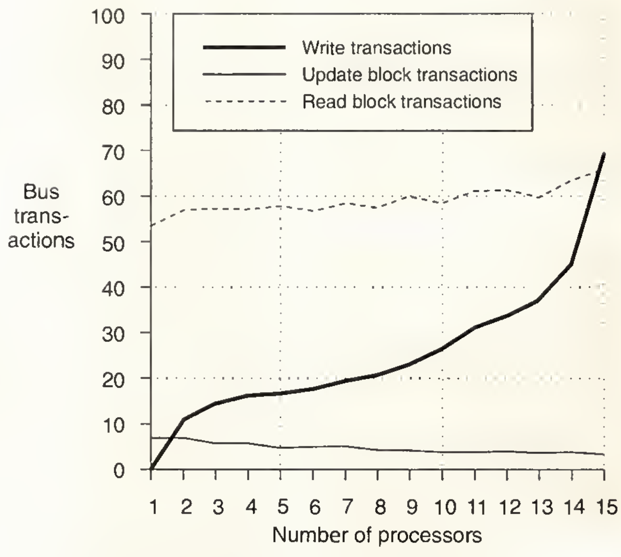


Figure 10. Write, update block, and read block transactions for 1,000 memory operations (for the RST protocol).

decrease in the number of update block transactions. As Figure 9 shows, Firefly uses a smaller number of update block transactions than either RST or Dragon because, unlike these two protocols, Firefly never requires an update block transaction upon the replacement of a shared copy.

RST behaves like Firefly in that it minimizes the number of shared copies needing update block transactions. In particular, RST never changes the copy state from shared-clean to shared-dirty during a write operation while Dragon does. Every cache uses a small number of these bus transactions (see Figure 10), which, therefore, are a low contributing factor to system performance.

The decrease in the perfomance of Firefly as compared with RST results from the:

- use of a low-pass filter to solve the wired-Or glitch problem. We assumed that multislave transactions were 25-ns slower than single-slave transactions.[14]
- time lost by several slaves during a read block transaction involving a shared copy for putting the data on the bus. Note that each cache executes activities pertinent to the processor operation and the bus transaction in parallel. However, we must serialize a number of phases of these two activities to guarantee copy consistency or because they involve internal shared structures. The fact that all the caches with a copy must put the data on the bus increases the need to serialize the internal activities of the caches involved. Indeed, the number of times that portions of potentially independent activities internal to the cache must be serialized is higher in Firefly than in RST (in which only one cache puts the data on the bus). Furthermore, the occurrence of more frequent conditions requiring the sequential execution of portions of activities implies a longer time for either internal activity to perform its task.

The data buffer in the shared memory allows Firefly to match the same system performance of Dragon. Q. Yang and others[34] reached the same conclusion in a performance analysis of cache coherence protocols for multiprocessors with a packet-switched shared bus. In such systems, a processor sends a request packet containing the desired operation and releases the bus without waiting for the completion of the operation. Once the shared memory finishes the requested operation, it sends the response packet back. In the case of a write transaction, our multiprocessor model has the same behavior as multiprocessors with a packed-switched shared bus.

General considerations. Detection of system events and cache conditions that contribute to system performance was the main aim of the simulation phase. Figure 10 shows that the number of write transactions needed to execute 1,000 memory operations increases as the number of processors increases, whereas the number of update block transactions decreases. Both variations are due to the increase of the memory references to the shared copies as shown in Figure 11.

Both the tendency of actual sharing and the fact that every write operation involving a shared copy requires a write transaction explain the curve of the write transactions. In the same way, we can explain the decrease of the update block transactions versus the number of processors. The replacement of a shared copy on which one or more write operations have been executed does not always require an update block transaction. However, an update block transaction is always needed for the replacement of a private copy involved in write operations.

In conclusion, the rapid drop in the performance of the multiprocessor that includes cache memories is partly the result of the increase in shared copies. The increase of shared copies modifies the behavior of every cache from copy-back to write-through when the number of processors increases. By a detailed analysis of the simulation results, we have found that certain events create shared copies from memory blocks belonging to private areas of processes. Actually, the increase in shared copies is mainly due to the migration of processes from one processor to another and the execution of kernel and I/O routines on different processors.

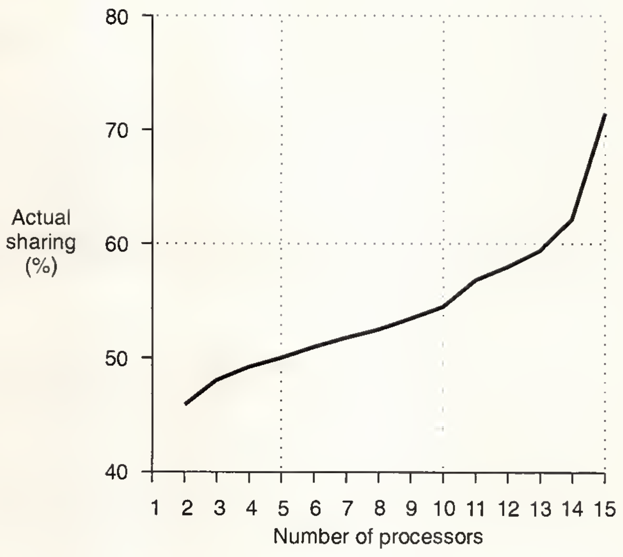


Figure 11. Percentage of operations involving a shared copy (in the case of the RST protocol and 10 percent of references to shared data).

RST cache design

Our multiprocessor prototype features a set of processors using a shared memory and I/O interfaces by means of a common bus (see Figure 12). Each processor has access to the local resources (32 Kbytes of RAM; 128 Kbytes of erasable, programmable read-only memory; RS-232C serial I/O port; and a 16-bit programmable counters/timers chip). It also includes a Clipper module,[16, 17] and an RST cache of 256 Kbytes. We can only use the local EPROM and RAM and the local serial I/O and counter/timer devices when the Clipper operates in the supervisor mode. In our prototype, we use these memories and devices to initialize the processor and carry out debugging[35-37] and testing functions.

The RST cache operates on a memory space of 2^{28} bytes by using a portion of the Clipper module's real address. The 32-bit Clipper synchronous bus connects the cache to the Clipper module. The 32-bit common bus connects the cache to the shared memory. RST is a two-way, set-associative cache with 8K sets; each includes two blocks (labeled as 0 and 1). Each cache block consists of a 16-byte data field, an 11-bit tag field (used to store bits A_{17} to A_{27} of the address of the corresponding memory block), and a 3-bit status field, which contains the

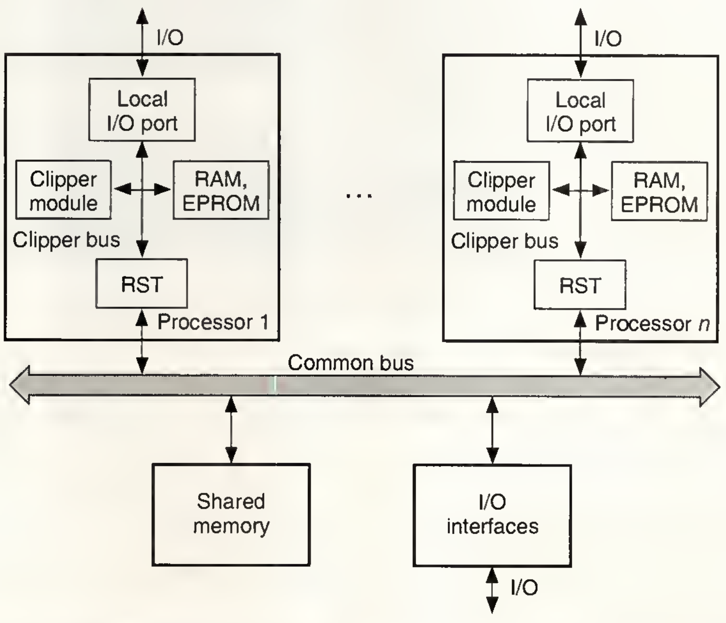


Figure 12. Architecture of the multiprocessor system based on the Clipper module.

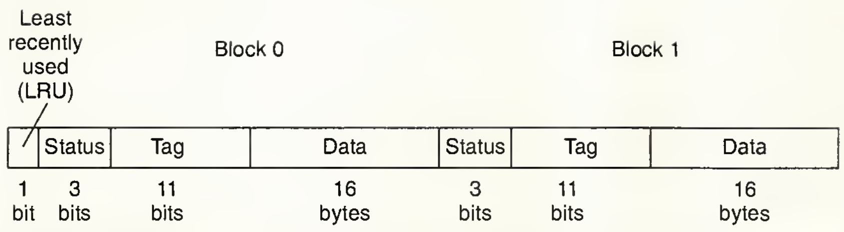


Figure 13. Configuration of a set.

state of the copy (see Figure 13). Finally, a 1-bit, least recently used (LRU) field is associated with each set; this field specifies the cache block (of the set) that is least recently used by the processor at any point in time.

Within the block, the processor can work on 1, 2, 4, or 16 bytes. The RST cache translates a memory address supplied by the processor into a cache memory address as follows:

- A_0 to A_3 is the byte address within the block,
- A_4 to A_{16} is the set address, and
- A_{17} to A_{27} is the address tag.

A_{28} to A_{31} are not used. During a processor operation, the address tag is compared with the tag field of both the cache blocks selected by the set address. When one of the two comparators signals a match with a valid copy, a cache hit results. When there is no match, we have a cache miss (see Figure 14).

The RST consists of the following:

- an operation manager, which performs all the actions relevant to the operations requested by the CPU;
- a watchdog, which maintains the consistency of the copies by monitoring the common bus transactions (see Figure 15);
- a switching and arbitration unit, which allows both the operation manager and the watchdog to operate on the block fields;
- data array, which implements the data fields of the cache blocks;
- LRU, tag, and status array, which implement the LRU fields

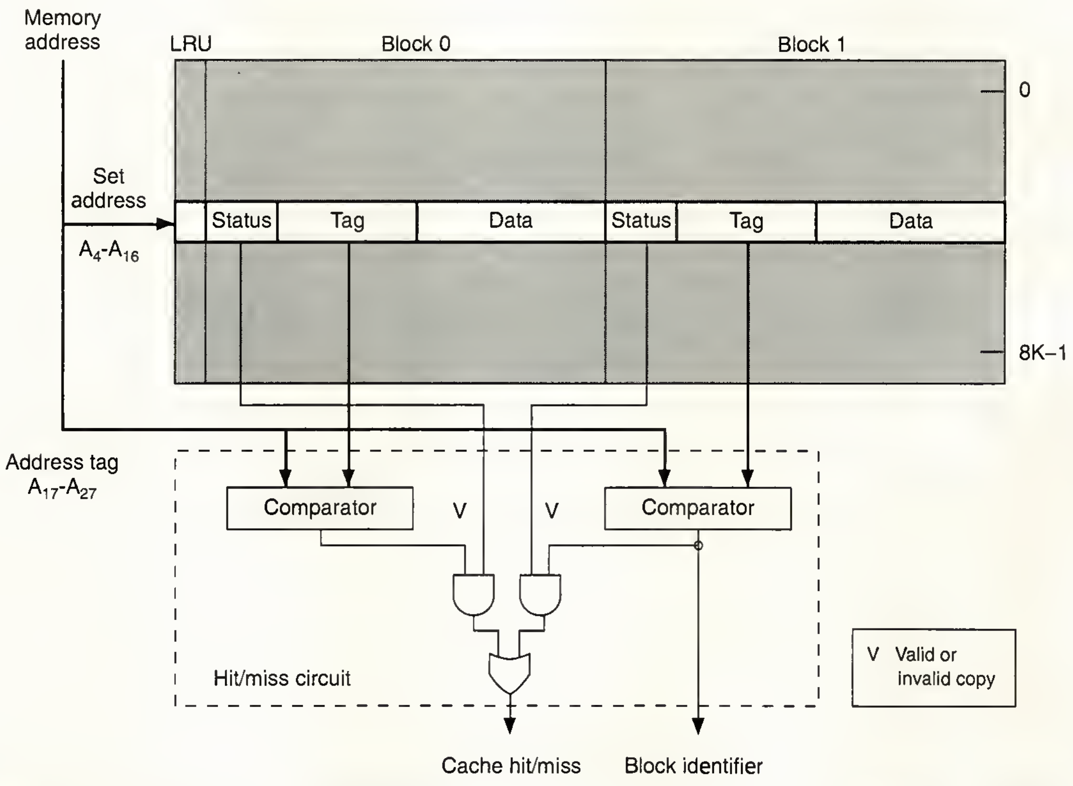


Figure 14. Cache access mechanism.

of sets and the tag and status fields of the cache blocks (used during local processor operations); and

- a copy of the tag and status array (used during bus transactions).

The tag and status fields are duplicated, allowing the operation manager and watchdog modules to operate in parallel. These two modules work sequentially only when both of them must access the data array or when one must modify the tag field, status field, or both. The operation manager and watchdog operate on the common bus through two common bus interface modules. The operation manager uses one module to operate on the common bus as a master unit for performing bus transactions; the watchdog uses the other module to operate on the common bus as a listening cache.

Moreover, the cache includes a write-operation buffer and a data buffer. When a local processor requires a write operation, the cache stores the relevant operation in the write-operation buffer and signals the completion of the operation. Then the cache executes the operation held in the buffer. As a result, the processor executes every write operation without wait cycles, apart from the possible time interval needed to complete the previous write operation.

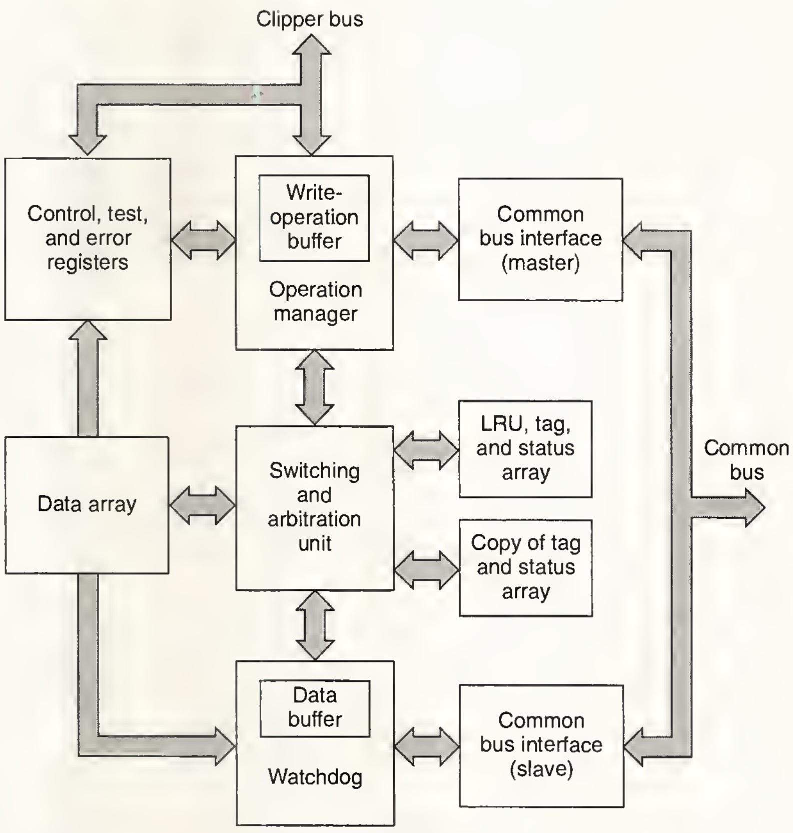

Clipper bus
Control, test, and error registers
Write-operation buffer
Operation manager
Common bus interface (master)
LRU, tag, and status array
Common bus
Data array
Switching and arbitration unit
Copy of tag and status array
Data buffer
Watchdog
Common bus interface (slave)

Figure 15. Diagram of a cache block.

In the same way, during a write transaction, the watchdog temporarily stores the data in the data buffer if it must update the copy involved. Therefore, if a processor operation involves the data array, the watchdog updates the data field of the cache block as soon as possible.

The cache also includes a mechanism that detects bus and cache errors. In the case of an error, the cache records the information pertinent to the failure in an error register and sends an interrupt request to the local processor. Finally, a control register makes it possible to enable and initialize the cache, whereas a test register is used to support the testing of the cache memory.

Main design problems. In the implementation of the RST coherence protocol we coped with certain problems arising from the particular architecture of the multiprocessor prototype and the features of the Clipper microprocessor.

Cache-to-cache transfer bus cycle. During a read block transaction, the RST protocol establishes that a listening cache having a dirty copy of the memory block involved in the bus transaction must take the place of the shared memory. For this reason, the common bus includes a special bus line, called dismem. This line, when active, disables the shared memory. In particular, during a read block transaction, a listening cache having a dirty copy of the involved memory block activates the dismem line before the end of the addressing phase of the bus transaction. Then the listening cache substitutes the shared memory by supplying the up-to-date copy of the memory block.

Bus transactions required by DMA devices. Certain I/O interfaces and communication subsystems can operate on the common bus as DMA devices. Besides read block and write transactions, a DMA device can use the read transaction that reads a byte, word, or double word from the shared memory. It can also use a burst-write transaction that transfers four double words to the shared memory. Therefore, we must also guarantee the consistency of the cached copies during a burst-write transaction without slowing down the DMA operation. We have adopted this simple solution: The watchdog invalidates the cached copy involved in the burst transaction by writing the invalid state in the status field of the pertinent cache block.

Consistency of the caches within the Clipper module. The Clipper module includes two 4-Kbyte, two-way, set-associative caches (one for data and one for instructions) with a 16-byte block length. Therefore, other copies of the memory blocks might be in these caches. Note that

- the Clipper caches cannot distinguish between private and shared copies, and
- RST cannot detect whether the Clipper caches have a copy of a memory block. (When the Clipper caches have a copy of a memory block, the local RST may or may not have a copy.)

For these reasons, it is necessary to prevent the existence of shared copies in the Clipper caches. We solve the problem via software. We programmed the Clipper caches to operate in the copy-back mode on the memory pages related to private data and in the noncacheable mode on the memory pages related to shared data. (In the latter mode, all operations are executed in the memory without holding a copy in the cache.) In addition, the Clipper's data cache is flushed on each context switch.

Implementation details. The operation manager and watchdog include fast combinatorial and registered programmable logic devices with a low-propagation-delay time (AmPAL 16L8-B, AmPAL 16R4-B, AmPAL 16R6-B, and AmPAL 16R8-B[38] produced by the Advanced Micro Devices), whereas fast-transistor-transistor-logic chips implement simple logic function and bus drivers.

Data array. Two 128-Kbyte memory banks obtain the data fields. Each memory bank employs 16 Cypress CY7C164 chips[39] (16K × 4 static RAM with a 25-ns access time).

LRU, tag, and status array. The LRU, tag, and status fields implement as two 16K × 16 memory banks (each bank employs four Cypress CY7C164 16K × 4 fast RAM chips). One memory bank implements the LRU field of the sets, and the tag and status fields of the number 0 blocks. The other memory bank implements the tag and status fields of the number 1 blocks.

The copy of tag and status array. In the same way, the copy of the tag and the status fields is implemented as two 16K × 16 memory banks.

The switching and arbitration unit. This unit includes two circuits generating hit/miss signals (one for the operation manager and the other for the watchdog unit), an LRU circuit that automatically updates the LRU bit, and an arbitration unit.

Each hit/miss circuit contains two 11-bit comparators, one for block 0 and one for block 1 (see Figure 14). Each comparator compares the 11-bit tag field of the memory block address with the 11-bit tag field of the selected cache block. When one of the two comparators reveals the match and the relative state is valid, the circuit activates the hit/miss signal. The output of the comparator operating on cache block 1 is the block identifier signal, and its value identifies the block (in the set) containing the copy. This signal selects the memory bank of the data array involved in the operation.

The LRU circuit only functions during operations when required by the operation manager. It writes into the involved LRU bit the negated value of the block identifier signal in the case of a hit condition. Otherwise, in the case of a miss condition, the circuit writes the old negated value of the LRU bit.

Finally, the arbitration unit accomplishes the following activities. It:

- arbitrates the access to the data, tag, and status fields. It forces the sequential execution of the operations required by the operation manager and watchdog when both must access the data fields or when one of them must modify a tag field, status field, or both;
- allows the operation manager to update both copies of a tag and/or a status field at the same time; and
- allows the watchdog to update both copies of a status field at the same time.

The arbitration unit contains an arbiter and a number of line drivers and bus transceivers. The arbiter uses a static priority scheme: For simultaneous requests it serves the watchdog unit first.

THE PRINCIPAL AIM IN THE RST CACHE DESIGN is to minimize the requirements of each processor on the common bus—the bottleneck of multiprocessor systems. We achieved this aim by using a two-way, set-associative cache managed in copy-back mode. A watchdog module guarantees consistency between the multiple copies of the same memory block by updating its possible copy when it intercepts a bus transaction involving the copy itself.

We derived the coherence protocol from the Dragon by eliminating the state transitions occurring in the case of a write operation on a shared copy. Also, this protocol operates like the Firefly by updating the shared memory on each write transaction without the drawbacks of the Firefly.

The simulation results show that the

- global system power of the RST cache design performs better on the average than that of the Firefly and Dragon protocols;
- performance depends on the percentage of write operations working on shared copies and, consequently, depends on the actual sharing value; and
- actual sharing is greater than the frequency of references to shared data. Certain events (such as process migration and execution of I/O interrupt routines on different processors) create a large number of shared copies derived from memory blocks belonging to private areas of processes.

From the results of this project, we have started to design a new coherence protocol aimed at reducing the actual sharing.

Acknowledgments

The Ministero dell' Università e della Ricerca Scientifica e Tecnologica, Italy partially supported this research. Many people contributed to the RST design. In particular, I heartily thank S. Cinquini for his help in the development phase and M. Favilla and G. Neri for their contributions during the simulation phase. I also thank the referees for their assistance regarding improvements. Finally, thanks to a number of researchers from Olivetti in Pisa for their valuable and helpful discussions.

References

1. P.H. Enslow, "Multiprocessor Organization," *ACM Computing Surveys*, Vol. 9, No. 1, Mar. 1977, pp. 103-129.
2. M. Satyanarayanan, *Multiprocessors: A Comparative Study*, Prentice Hall, Englewood Cliffs, N.J., 1980.
3. A.K. Jones and P. Schwarz, "Experience Using Multiprocessor Systems: A Status Report," *ACM Computing Surveys*, Vol. 12, No. 2, June 1980, pp. 121-165.
4. P. Corsini and C.A. Prete, "Architecture of the Mu TEAM System," *IEE Proc.*, IEE, Stevenage, Herts, England, Vol. 134, Pt. E, No. 5, Sept. 1987, pp. 217-227.
5. K. Hwang and F.A. Brigg, *Computer Architecture and Parallel Processing*, McGraw-Hill Series in *Computer Organization and Architecture*, McGraw-Hill, New York, 1984.
6. A.J. Smith, "Cache Memories," *ACM Computing Surveys*, Vol. 14, No. 3, Sept. 1982, pp. 473-530.
7. P.J. Denning, "On Modeling Program Behaviour," *Proc. Spring Joint Computer Conf.*, AFIPS Press, Arlington, Va., Vol. 40, 1972, pp. 937-944.
8. J.R. Goodman, "Cache Memory Optimization to Reduce Processor/Memory Traffic," *J. of VLSI and Computer Systems*, Vol. 2, No. 1-2, 1987, pp. 61-86.
9. J. Archibald and J.-L. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," *ACM Trans. Computer Systems*, Vol. 4, No. 4, Nov. 1986, pp. 273-298.
10. L.M. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Trans. Computers*, Vol. C-27, No. 12, Dec. 1978, pp. 1112-1118.
11. M. Dubois and F. Briggs, "Effects of Cache Coherency in Multiprocessors," *IEEE Trans. Computers*, Vol. C-31, No. 11, Nov. 1982, pp. 1083-1099.
12. R. Katz et al., "Implementing a Cache Consistency Protocol," *Proc. 12th Int'l Symp. Computer Architecture*, IEEE, New York, 1985, pp. 276-283.
13. M. Papamarcos and J. Patel, "A Low Overhead Coherence Solution for Multiprocessors with Private Cache Memories," *Proc. 11th Int'l Symp. Computer Architecture*, 1984, pp. 348-354.
14. P. Sweazey and A.J. Smith, "A Class of Compatible Cache Consistency Protocols and Their Support by the IEEE Futurebus," *Proc. 13th Int'l Symp. Computer Architecture*, 1986, pp. 414-423.
15. W.C. Yen and K.S. Fu, "Coherence Problem in a Multicache System," *Proc. Int'l Conf. Parallel Processing*, IEEE, New York, 1982, pp. 332-339.
16. *Clipper Module 32-Bit Microprocessor User's Manual*, Prentice Hall, Englewood Cliffs, N.J., 1987.
17. C.B. Hunter, "Introduction to the Clipper Architecture," *IEEE Micro*, Vol. 7, No. 4., Aug. 1987, pp. 6-26.
18. L. Monier and P. Sindhu, "The Architecture of the Dragon," *Proc. 13th IEEE Int'l Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1985, pp. 118-121.
19. *ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic*, IEEE CS Press, 1985.
20. C.H. Hoogendoorn, "Reduction of Memory Interference in Multiprocessor Systems," *Proc. 4th Ann. Symp. Computer Architecture*, IEEE, New York, 1977, pp. 179-183.
21. J. Archibald and J.-L. Baer, "An Economical Solution to the Cache Coherence Problem," *Proc. 11th Int'l Symp. Computer Architecture*, 1984, pp. 355-362.
22. C.K. Tang, "Cache System Design in the Tightly Coupled Multiprocessor System," *Proc. AFIPS Nat'l Computer Conf.*, Vol. 45, 1976, pp. 749-753.
23. S. Thakkar, P. Gifford, and G. Fielland, "The Balance Multiprocessor System," *IEEE Micro*, Vol. 8, No. 1, Feb. 1988, pp. 57-59.
24. G.J. Myers, *Advances in Computer Architecture*, 2nd ed., Wiley-Interscience, New York, 1982.
25. E.A. Feustel, "On the Advantages of Tagged Architecture," *IEEE Trans. Computers*, Vol. C-22, No. 7, July 1973, pp. 644-656.
26. J.R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," *Proc. 10th Int'l Symp. Computer Architecture*, 1983, pp. 124-131.
27. C.P. Thacker, L.C. Stewart, and E.H. Satterthwaite, "Firefly: A Multiprocessor Workstation," *IEEE Trans. Computers*, Vol. C-37, No. 8, Aug. 1988, pp. 909-920.
28. S.J. Frank, "Tightly Coupled Multiprocessor System Speeds Memory-Access Times," *Electronics*, Vol. 57, No. 1, Jan. 12, 1984, pp. 164-169.
29. S. Cinquini and C.A. Prete, "An Example of Cache Memory Design for Multiprocessor Systems Based on Clipper Fairchild Microprocessor," *Proc. 3rd Int'l Symp. Computer and Information Sciences*, Nova Science Publishers, New York, Oct./Nov. 1988, pp. 597-604.
30. D.M. Ritchie and K. Thompson, "The Unix Time-Sharing System," *Commun. of the ACM*, Vol. 17, No. 7, July 1974, pp. 365-375.
31. J.S. Quaterman, A. Silberschatz, and J.L. Peterson, "4.2BSD and 4.3BSD as Examples of the Unix System," *ACM Computing Surveys*, Vol. 17, No. 4, Dec. 1985, pp. 379-418.
32. M.J. Bach and S.J. Buroff, "Multiprocessor Unix Operating Systems," *AT&T Bell Laboratories Technical J.*, Vol. 63, No. 8,

Oct. 1984, pp. 1733-1749.
33. P. Corsini, B. Lazzerini, and C.A. Prete, "A Kernel for a Multiprocessor System with Anonymous Processes," *Proc. Int'l Conf. Parallel Processing and Applications,* North-Holland, Amsterdam, Sept. 1987, pp. 71-78.
34. Q. Yang, L.N. Bhuyan, and B.-C. Liu, "Analysis and Comparison of Cache Coherence Protocols for a Packet-Switched Multiprocessor," *IEEE Trans. Computers,* Vol. 38, No. 8, Aug. 1989, pp. 1143-1153.
35. B. Lazzerini, L. Lopriore, and C.A. Prete, "A Programmable Debugging Aid for Real-Time Software Development," *IEEE Micro,* Vol. 6, No. 3, June 1986, pp. 34-42.
36. P. Corsini and C.A. Prete, "Multibug: Interactive Debugging in Distributing Systems," *IEEE Micro,* Vol. 6, No. 3, June 1986, pp. 26-33.
37. B. Lazzerini and C.A. Prete, "Event-Driven Debugging for Distributed Software," *Microprocessors and Microsystems,* Vol. 12, No. 1, Jan./Feb. 1988, pp. 33-39.
38. *PAL Device Data Book,* Advanced Micro Devices, Inc., 1988.
39. *CMOS Data Book,* Cypress Semiconductor Corp., San Jose, Calif., Jan. 1986.

Cosimo A. Prete is a research fellow at the Institute of Electronics and Telecommunications of the University of Pisa. He is involved in the Italian National Research Council's Nonconventional Parallel Systems project, which is conducting comparative analysis and performance evaluation of operating systems and programming environments for nonconventional parallel systems. His main interests include multiprocessor organization, cache memories, and software development methodologies.

Prete holds a degree in electronic engineering and a PhD (Dottore di Ricerca) from the University of Pisa, Italy.

Address questions concerning this article to Cosimo A. Prete, Università di Pisa, Dipartimento di Ingegneria dell' Informazione: Elettronica, Informatica e Telecommunicazioni, 56126 Pisa, Via Diotisalvi, 2, Italy; e-mail: prete@mv3500.eit.unipi.it.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate numbers on the Reader Service Card.

Low 153 Medium 154 High 155

Communication latency

continued from p. 15

nodes in a hypercube if we know the dimension of the hypercube, or $\log_2(N)$. The same factor in a 2D grid is $O(\sqrt{N})$, which increases faster than $\log_2(N)$.

Recall that the communication latency is dependent on the channel width, distance (number of hops), and size of the message. The network latency in the wormhole model precisely equals the time it takes the head of a message to enter the network at the source and the tail to emerge at the destination:

$$T_{lat} = t_d H + (K/B) \tag{5}$$

Here, T_d is the delay of the individual routing nodes encountered on the path, H is the number of hops needed in passing messages, and K/B is the time required for a message of size K to pass through the channels of bandwidth B.

In lower dimensional hypercube topology, the number of hops increases, but so does the channel width. The optimization to minimize latency simply minimizes Equation 5. In this equation, higher dimensional networks reduce the first term at the expense of the second, while lower dimensional networks reduce the second term at the expense of the first. The 2D grid has $O(\sqrt{N})$ times more wires per channel for a fixed number N of nodes than an equivalent N-node topology. The following numerical comparisons indicate the advantage of lower latency in the 2D grid network. Assume the routing delay T_d in both the hypercube and 2D grid networks is identical. We can express the time needed to send a message with K bytes between a pair of nodes in the maximum distance $\log_2(N)$ in the hypercube with N nodes as:

$$T_{cube} = T_d \log_2(N) + (K/B) \tag{6}$$

We express the same timing factor in the 2D grid network of N nodes to send a K-size message differently, since the bandwidth in each channel is $O(\sqrt{N})$ times wider, and the maximum distance $O(\sqrt{N})$ is also a faster-increasing function:

$$T_{\text{grid}} = T_d O(\sqrt{N}) + \frac{K}{O(\sqrt{N})B} \tag{7}$$

We derive the ratio of T_{cube}/T_{grid} by

$$R = O(\sqrt{N}) \frac{T_d \log(N) + T_c}{T_d N + T_c} \tag{8}$$

where $T_c = K/B$.

The second term K/B of Equation 5 dominates the network latency for all but very short messages in the second-generation multicomputers. For example, in one implementation conducted by the California Institute of Technology,[16] the routing delay in one node T_d was 80 nanoseconds. Even the fast bandwidth of the Ametek 2010—B = 20 Mbytes/s needed to transfer a 160-byte message—would take 8 microseconds, which is 100 times longer than T_d. When K is reasonably large, T_d may be ignored, and the ratio R of Equation 8 is $O(\sqrt{N})$. That is, the communication latency in a 2D grid network may be reduced up to $O(\sqrt{N})$ times over a hypercube network.

In summary, given a constant bisection width, the 2D grid network produces lower latency and higher throughput than a higher dimensional hypercube. Mainly, fewer channels contribute to the bisection, which permits each channel to be made wider. On the other hand, the throughput is bounded by allowing more channels crossing the bisection in a higher dimensional hypercube.

A transputer is a good candidate for constructing a 2D grid network.

The Topology 1000 provides hardware reconfigurability of the network topology under software control through the use of the Inmos C004 link crossbar adapter. Thus, a user can define an interprocessor communication topology, and the hardware and software can implement it. Since each transputer has four links to connect with other transputers in the network, a transputer becomes a good candidate for constructing a 2D grid network. We can achieve full connectivity or high connectivity in a lower dimensional topology with a small number of nodes and construct a 4D hypercube by connecting 16 transputers properly.

However, we cannot build higher dimensional hypercubes out of transputers exclusively since they are limited to four links per node, and hypercubes of five or more dimensions require five or more links per node. Such topologies are possible with the addition of hardware link switches such as the Inmos C004 crossbar adapter used in the Topologix system. Performance losses occur with the use of such switches, however.

The experiment

A distributed-memory multicomputer is a collection of processors or nodes connected by a communication network. Thus, the basic communication timing test for distributed-memory multicomputers requires measurement of the time required to transmit a message packet from one node to its nearest neighbor. This test, also known as an echo test, directs a test node to send a message to an echo node that is directly connected to the test node. The echo node receives the message and sends it back to the test node. We can express the interprocessor communication time required to transmit a message between two directly connected nodes as:

$$T_{comm} = \alpha + \beta K$$

where K is the number of bytes contained in the message. Here, α equals the overhead or the start-up time for sending a packet in microseconds, and β equals the bandwidth of the communication channel (microseconds/byte). The experiment used different sizes of message packets and a least square fit to approximate α and β. Table 2 reproduced from Zhang and Beguelin[17] lists the αs and βs of the five types of multicomputers.

Table 2. Alphas and betas (in microseconds/byte) for one-hop communication.

Multiprocessor	α	β
iPSC/1	893	1.51
iPSC/2	349	2.30×10^{-1}
Ncube/10	447	2.40
Ametek 2010	168	1.01×10^{-1}
Topology 1000	215	1.02×10^{-1}

Since multiple-hop communications occur more often in most applications on a multiprocessor system, the one-hop communication measurements do not let us sufficiently evaluate the performance of the interprocessor communication. For this reason, we[17] constructed a comprehensive experiment to measure the overall communication performance on a multiprocessor system for a given topology network.

In the experiment, a test node sent n messages to and received n messages from all nodes in the network. We measured the time it took for a test node to send a message to every node in the network and return. We repeated this process m times and continued the whole process until every node had become the test node. We obtained the average communication time in the network from the p timing measures, where p is number of processors in the network. We chose the message size from a minimum of 1 byte to a maximum of 8 Kbytes. The communication distances in this experiment range from a minimum zero hop (a node to itself) to a maximum H_{max} hops. $H_{max} = n$, for an n-dimensional hypercube topology, and $H_{max} = O(\sqrt{N})$, for an N-node 2D grid topology.

Figure 2 charts the average communication time for different message sizes on different types of multiprocessors. The iPSC/1, iPSC/2, and Topology 1000 have a hypercube topology, and the Ametek 2010 has 2D grid topology. The results of the experiment showed that communication timing differences are very close to the results predicted earlier by the latency models. For example, Equation 4 predicted the iPSC/1 and iPSC/2 communication latency ratio for a 16-node system to be 12.8. The experiment's results in Figure 2 also show that the timing ratio was more than 10.

To show that the communication latency of the wormhole model exhibits little sensitivity to message distance, we conducted another experiment on the five types of multicomputers. In this experiment we fixed the message size, let the communication time become the function of the distance, and set the number of hops as *H*. We ran this experiment with message-packet sizes of 1 Kbytes to 8 Kbytes and used the average timing value from eight runs as the measure to cover a wide range of message sizes. Figure 3 describes this timing function based on the experimental data.

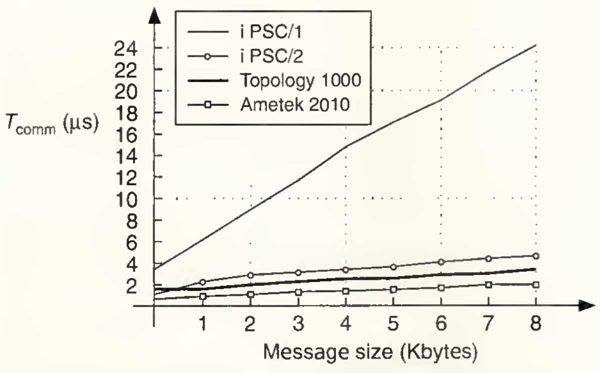


Figure 2. Average multihop communication time on the Intel, Topologix, and Ametek multiprocessors.

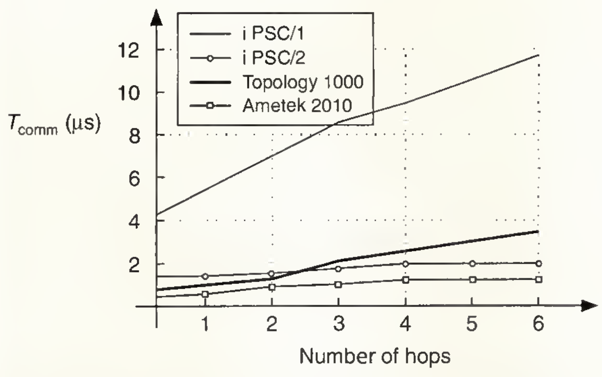


Figure 3. Average communication time with different hops on the same multiprocessors.

The experiment's results clearly show the performance difference of the interprocessor communication between the first-generation multicomputer systems and the second-generation distributed multiprocessor systems. The traditional store-and-forward technique for interprocessor communication greatly limits the communication speed among the processors. In addition, the processors of the first-generation multiprocessing systems are not very powerful, which is another major reason communication proceeds slowly in these systems.

To transfer a message in a store-and-forward network, such as the iPSC/1 or the Ncube/10, the processor must move each byte of data through its own memory, thus consuming both storage bandwidth and computing cycles in the routing nodes. The Intel iPSC/2 uses more powerful processors and, more importantly, uses direct switches as the interprocessor connections. Thus, the communication performance is greatly improved over that of the iPSC/1 and the Ncube/10.

The Topology 1000's high-performance interprocessor communication occurs especially when the number of processors in the network is not very large and the message size is not too small. The four links of each transputer, which may create more hops and a low number of direct connections for a large number of transputer networks, limit the inter-transputer connectivity. The DMA data links in the multiple-transputer system play an important role in transferring data at high speeds. However, as the graph in Figure 3 shows, the communication latency of the Topology 1000 is more sensitive than are the iPSC/2 and Ametek 2010 when the number of hops increases. The Topology 1000 uses the store-and-forward model, after all. We obtained the timing results from a 16-node hypercube network on both the iPSC system and the Topology 1000. Finally, the point-to-point communication established on the Ametek 2010, which contains a powerful mesh routing chip on each node, produces the best interprocessor communication performance among the five multiprocessor architectures.

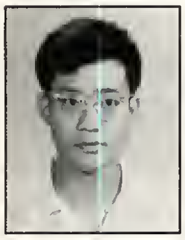
THE WORMHOLE ROUTING MODEL greatly reduces communication latency and is no longer sensitive to the distance involved in passing messages. In addition, the high-data bandwidth and high-speed nodes of the second-generation multicomputers such as the iPSC/2 and Ametek 2010 increase communication speed. The Topology 1000 interprocessor communication may perform at a rate similar to that of the iPSC/2 and Ametek 2010 on a medium-size network since the system takes advantage of the high-speed transputer data links. The 2D grid topology is a more efficient structure than a higher dimensional hypercube topology in terms of reducing communication latency, as long as the routing delay in each node is small, such as the one in the second-generation multicomputers.

Acknowledgments

The National Science Foundation under grants CCR-9008991 and CCR-9047481 partially supported this research.

References

1. C. Moler, "Matrix Computation on Distributed-Memory Multiprocessors," *SIAM Proc. Hypercube Multiprocessors*, Soc. Industrial and Applied Mathematics, Philadelphia, 1986, pp. 181-195.
2. X. Zhang, R. Byrd, and R. Schnabel, "Solving Nonlinear Block-Bordered Circuit Equations on Hypercube Multicomputers," *Proc. Fourth Conf. Hypercubes, Concurrent Computers, and Applications*, Vol. I, 1989, pp. 701-707.
3. A. Beguelin and D. Vasicek, "Communication Between Nodes of a Hypercube," *SIAM Proc. Hypercube Multiprocessors*, 1987, pp. 162-168.
4. D.A. Reed and R.M. Fujimoto, *Multicomputer Network: Message-Based Parallel Processing*, MIT Press, Cambridge, Mass., 1987.
5. D.C. Grunwald and D.A. Reed, "Benchmarking Hypercube Hardware and Software," *SIAM Proc. Hypercube Multiprocessors*, 1987, pp. 169-175.
6. Y. Saad and M.H. Schultz, "Data Communication in Hypercubes," *J. Parallel Distributed Computing*, Vol. 6, 1989, pp. 115-135.
7. C.L. Seitz, "The Cosmic Cube," *Comm. ACM*, Vol. 28, No. 1, 1985, pp. 22-33.
8. *iPSC User's Guide*, No. 17455-3, Intel Corp., Portland, Ore., 1985.
9. P. Pierce, "The NX/2 Operating System," *Proc. Third Conf. Hypercube Concurrent Computers and Applications*, ACM Press, 1988, pp. 384-390.
10. *Ncube Handbook, Version 1.1*, Ncube Corp., Beaverton, Ore., 1986.
11. *The Transputer Data Book*, Inmos Corp., Bristol, UK, 1989.
12. *The Transputer Application's Notebook: Architecture and Software*, Inmos Corp., 1989.
13. *The Transputer Application's Notebook: Systems and Performance*, Inmos Corp., 1989.
14. A.S. Tanenbaum, *Computer Network*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.
15. P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, Vol. 13, 1979, pp. 267-286.
16. W.C. Athas and C.L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *Computer*, Vol. 21, No. 8, 1988, pp. 9-24.
17. X. Zhang and A. Beguelin, "Interprocessor Communication Performance on Different Types of Multicomputers," *Intelligent Distributed Processing*, R. Ammar, ed., ACTA Press, Anaheim, Calif., 1989, pp. 73-76.



Xiaodong Zhang is an assistant professor of computer science at the University of Texas at San Antonio and holds a visiting faculty position at the Center for Research on Parallel Computation at Rice University. Earlier, he had worked as a member of the technical staff for Topologix Inc., Denver. His research interests lie primarily in the areas of parallel and distributed computation, parallel system performance evaluation and VLSI simulation, and numerical analysis of nonlinear equations and optimization problems.

Zhang received the BS degree in electrical engineering from Beijing Polytechnic University and the MS and PhD degrees in computer science from the University of Colorado at Boulder. He is a member of the IEEE Computer Society, the Association of Computing Machinery, and the Society for Industrial and Applied Mathematics.

Address questions concerning this article to the author at the Division of Mathematics and Computer Science, University of Texas at San Antonio, San Antonio, TX 78285-0664; or via Internet at zhang@ringer.cs.utsa.edu.

Reader Interest Survey

Indicate your interest in the article by circling the appropriate number on the Reader Service Card.

Low 150 Medium 151 High 152

Petri-net simulation

continued from p. 23

- a TMS32020 CPU (see Figure 4);
- a 16-Kbyte data RAM expandable to 64 Kbytes;
- a 1-Kbyte, erasable, programmable read-only memory containing the simulation program and the communication routines between the board and the IBM PC;
- an 8-Kbyte program RAM, required for storage of the procedures associated with the PN transitions; and
- a communication interface that allows data exchange between the DSP board and the IBM PC.

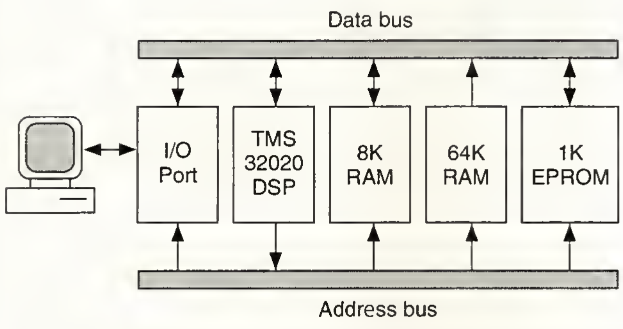


Figure 4. Block diagram of the hardware architecture for the TMS32020 implementation.

An easy-to-use interface written in Pascal allows users to input the data describing the PN to be simulated. Such data, initially stored in subsidiary files on the hard disk, transfer to the data RAM of the DSP board.

Users can write any procedures associated with the PN transitions in DSP assembly or Pascal (by using a suitable compiler for TMS32020). The first option obviously involves a good mastery of assembly language on the part of the user, and it provides an optimized code. The high-level language option, on the other hand, makes procedure implementation easy, but produces a redundant code. The maximum size we implemented in the simulator was 100 words per procedure. Then the simulation results, stored in a suitable RAM area, shift to output files.

Memory organization. The first problem we dealt with in defining the software architecture was data memory mapping. The CPU data memory consists of five-hundred-twelve 128-byte pages; the on-chip memory resides on pages 0 (locations 96-127; block B2), 4-5 (block B0), and 6-7 (block B1). We set the storage of all the information most frequently applied for, such as lists E and C, and some counters and user buffers to take advantage of the quick access to the on-chip data RAM. The storage of data describing the simulated PN—the PM and P matrices—starts from page 144.

```
LAB5 LAR 1,65
   SAR 1,FREE
   LAC *+                       ;extract the
   SACL ATTRAN                  ;transition
   LAC *+                       ;from the top
   SACL ATTIME                  ;of the list E,
LAB6 LAC *                      ;put it in
   SACL 65                      ;ATTRAN and
   LAC 81                       ;modify the
   SUBS ONE                     ;pointer to
   SACL 81                      ;the list E
   CALL FREMEM,*,2
   LT ATTRAN                    ;put the
   MPYK 12                      ;address of
   PAC                          ;the 1st field
   ADDS 88                      ;of ATTRAN in
   SACL INTRAN                  ;INTRAN
   LAR 1, INTRAN
   LARK 0, 10
   MAR *0+
   LAC *                        ;if the field
   AND MASK2                    ;"procedure" is
   BZ CC                        ;empty go to CC
   SACL TEMP1
   LALK 924                     ;otherwise
   LT TEMP1                     ;calculate the
   MPYK 100                     ;procedure
   APAC                         ;address and
   CALA                         ;call it
*Each procedure must be
*terminated with ARP=1
CC LAR 1, INTRAN
   LARK 4,4
CC1 LAC *+                      ;if the input=0
   BZ CC2,*,1                   ;then go to CC2
   SACL POSTO                   ;otherwise the
   CALL R1,*,2                  ;mark leaves
   LARP 4                       ;the input
   BANZ CC1,*-,1
CC2 LAR 1, INTRAN
   LARK 0,5
   MAR *0+
   LARK 4,4
CC3 LAC *+                      ;if output=0
   BZ T                         ;then go to T
   SACL POSTO                   ;otherwise
   CALL R2,*,2                  ;put the mark
   LARP 4                       ;in the output
   BANZ CC3,*-,1
T LAC 81                        ;if list E empty
   BZ GEST                      ;then go to GEST
   LAR 1,65                     ;otherwise
   SAR 1,FREE                   ;process the
   LAC *+                       ;transition on
   SACL ATTRAN                  ;the top
   LAC *+                       ;if it has the
   SUBS ATTIME                  ;same time of the
   BZ LAB6                      ;previous one
```

Figure 5. Listing of the firing mechanism module.

SUBSCRIPTION CARD READER-SERVICE CARD READER-SERVICE CARD

IEEE MICRO

April 1991 issue (card void after October 1991)

Name ______________________________

Title ______________________________

Company ______________________________

Address ______________________________

City ______________ State ______ Zip ______

Country ______________ Phone(______) ______

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150	151	152	180	181	182
153	154	155	183	184	185
156	157	158	186	187	188
159	160	161	189	190	191
162	163	164	192	193	194
165	166	167	195	196	197
168	169	170	198	199	200
171	172	173	201	202	203
174	175	176	204	205	206
177	178	179	207	208	209

Product Information 1
(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	
11	31	51	71	91	111	131	
12	32	52	72	92	112	132	
13	33	53	73	93	113	133	
14	34	54	74	94	114	134	
15	35	55	75	95	115	135	
16	36	56	76	96	116	136	
17	37	57	77	97	117	137	
18	38	58	78	98	118	138	
19	39	59	79	99	119	139	
20	40	60	80	100	120	140	

IEEE MICRO

April 1991 issue (card void after October 1991)

Name ______________________________

Title ______________________________

Company ______________________________

Address ______________________________

City ______________ State ______ Zip ______

Country ______________ Phone(______) ______

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150	151	152	180	181	182
153	154	155	183	184	185
156	157	158	186	187	188
159	160	161	189	190	191
162	163	164	192	193	194
165	166	167	195	196	197
168	169	170	198	199	200
171	172	173	201	202	203
174	175	176	204	205	206
177	178	179	207	208	209

Product Information 2
(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	
11	31	51	71	91	111	131	
12	32	52	72	92	112	132	
13	33	53	73	93	113	133	
14	34	54	74	94	114	134	
15	35	55	75	95	115	135	
16	36	56	76	96	116	136	
17	37	57	77	97	117	137	
18	38	58	78	98	118	138	
19	39	59	79	99	119	139	
20	40	60	80	100	120	140	

SUBSCRIBE TO IEEE MICRO

All the facts about today's chips and systems

❑ **YES**, sign me up!

If you are a member of the Computer Society or any other IEEE society, pay the member rate of only $21 for a year's subscription (six issues).

Society: ______________________________

IEEE membership no: ______________________________

Society members: Subscriptions are annualized. For orders submitted March through August, pay half the full-year rate ($10.50) for three bimonthly issues.

❑ **YES**, sign me up!

If you are a member of ACM, ACS, BCS, IEE (UK), IEEE but not a member of an IEEE society), IECEJ, IPSJ, NSPE, SCS, or other professional society, pay the sister-society rate of only $38 for a year's subscription (six issues).

Organization: ______________________________

Membership no: ______________________________

____
Full Signature Date

____
Name

____
Street

____
City

____
State/Country ZIP/Postal Code

❑ Payment enclosed *DC residents: add sales tax*

❑ Charge to ❑ Visa ❑ MasterCard ❑ American Express

Charge-card number

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Expiration date

_ _ _ _ _ _ _ _
Month *Year*

Prices valid through 12/31/91
4/91 MICRO

Charge orders also taken by phone:
(714) 821-8380 8 a.m. to 5 p.m. Pacific time
Circulation Dept.
10662 Los Vaqueros Circ., PO Box 3014
Los Alamitos, CA 90720-1264

Editorial comments

I liked: ______________________

____

____

I disliked: ______________________

____

____

I would like to see: ______________________

____

____

Reviewers Needed. *If interested, send professional data to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129, Torino, Italy.*

For reader-service inquiries, see other side.

PO Box is for reader-service cards only.

PLACE
POSTAGE
HERE

IEEE Micro
PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA

Editorial comments

I liked: ______________________

____

____

I disliked: ______________________

____

____

I would like to see: ______________________

____

____

Reviewers Needed. *If interested, send professional data to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129, Torino, Italy.*

For reader-service inquiries, see other side.

PO Box is for reader-service cards only.

PLACE
POSTAGE
HERE

IEEE Micro
PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 38 LOS ALAMITOS, CA

POSTAGE WILL BE PAID BY ADDRESSEE

IEEE COMPUTER SOCIETY
PO BOX 3014
LOS ALAMITOS CA 90720-9804
USA

Software program. The program essentially segments into two sections. The first section governs the reception of data from the PC, representing the simulated PN and the procedures associated with the transitions. The second section governs the actual simulation. The program initially scans the PM matrix to initialize the E list; then it performs a main loop—with calls for subroutines—to manage the simulation as shown in the Figure 3 and 4 flowcharts. The loop only ends when it finds list E empty or reaches the stop simulation time. In both cases the DSP transfers the results—represented by the 24 user buffers—to the PC, then it resets for a subsequent simulation session.

The main loop consists of three blocks. The first one, shown in Figure 5, performs the firing mechanism of the transition(s) that take the shortest amount of time It pops the firing transition(s) from list E (lines 1-12), points to the beginning of the associated procedure (if any), then executes the same (lines 13-29). The input places are reset (lines 32-39), and the output places are set (lines 40-49) when fulfilling the procedure. These steps repeat for all transitions that take the shortest amount of time. Lines 50-58 perform iteration control.

The second block (see Figure 6) manages the possible conflicts occurring during the PN evolution, according to the rules specified in the flowchart in Figure 3.

The third block checks whether the simulation time has expired or list E is empty.

The interesting routine Insup (Insert and Update) controls the ranked insertion of the enabled transitions into list E. The first block of the main loop calls off this routine when a token is put onto an output place of a fired transition. This marking completes the input configuration of a new nonconflicting transition. But the second block of the main loop recalls it when making a nondeterministic choice among several conflicting transitions.

Nondeterministic choice uses the short routine RND. This routine arbitrates a random number (0-199) associated with the enabled transition according to the formula:

$$R = (\text{RND mod } N) + 1$$

where N is the number of conflicting transitions and R is the relevant position of the selected transition in list C.

TMS32010 implementation. We implemented a reduced version of the simulator outlined here on a TMS32010 board. This board contains a CPU, a 4-Kbyte program RAM, and an interface for communicating with the PC.

We set aside a section of program memory for PN storage since the TMS32010 does not provide for an external data memory (the one-hundred-forty-four 16-bit words on chip memory pursue this target). The on-chip data memory for the TMS32020 operates as a working area for storage of the lists, counters, user information, and pointers to the procedures, as per the user's directions.

```
GEST LAC CONTA              ;if the list E
  BZ TT,*,2                 ;empty then go
  ZALH 81                   ;to TT
  BZ GM
  LAR 2,CONTA
  MAR *-,1
  LAR 1,TESTAA              ;AR1 points to
  LALK 870                  ;the list C
  SACL OLD                  ;head
G8 LAC *+,0,3
  SACL TEMP1                ;put in TEMP1
  LT TEMP1                  ;a transition t
  MPYK 12                   ;of list C
  PAC
  ADDS 88
  SACL TEMP2                ;TEMP2 points
  LAR 3,TEMP2               ;to the 1st
  LRLK 4,>FFFF              ;input of t
  LARK 0,4
G1 LAC *+,0,4
  BZ G2
  MAR *+,0
  BANZ G1,*-,3
G2 SAR 4,TEMP9
  LARP 3
  LAR 3,81
  MAR *-,0
  LAR 0,65
G7 LT *+
  SAR 0, TEMP3              ;TEMP3 points
  MPYK 12                   ;to the next
  PAC                       ;transition T
  ADDS 88                   ;of list E
  SACL TEMP4
  LAR 0, TEMP4
  LALK >FF04
  SACL TEMP6
  SACH TEMP5
G3 LAC *+
  BZ G4
  LAC TEMP5
  ADDS ONE
  SACL TEMP5
  SUBS TEMP6
  BNZ G3
G4 LAC TEMP5
  SACL TEMP6
  LAR 0, TEMP4
G9 LAC *+
  SACL TEMP7
  SAR 0, TEMP4
  LAR 4, TEMP9
G5 LAR 0, TEMP2             ;if t and T
  LAC *+,0,4                ;have a common
  SUBS TEMP7                ;input then go
  BZ G10                    ;to G10
  BANZ G5,*-,0
  LAR 0, TEMP6
  BANZ G6
  LAR 0, TEMP3
  LAR 0,*,3
```

Figure 6. Conflict management module listing (continued on next page).

```
    BANZ G7, *-, 0
    SAR 1,OLD
    LARP 1
G11 LAR 1,*,2
    BANZ G8, *-, 1
GM ZALH CONTA              ;if the list C
    BZ TT,*,2              ;empty go to TT
    SUBH ONE
    SACL TEMP1
    BZ G12
    CALL RND
    LAC RANDOM
    RPTK 15
    SUBC CONTA
    SACH TEMP1
G12 LAR 2, TESTAA
    LALK 870
    SACL OLD
    LAC TEMP1
    BZ G13
    LAR 0, TEMP1
    LARP 0
    MAR *-,2
G0 MAR *+
    SAR 2,OLD
    LAR 2,*,0
    BANZ GO,*-,2
G13 LAC *+                 ;extract the
    SACL ATTRAN            ;chosen
    LAC *-                 ;transition
    SAR 2,FREE             ;from list C
    LAR 2,OLD
    SACL *,0,1
    LAC CONTA
    SUBS ONE
    SACL CONTA
    CALL GIVEA
    LACK 11
    LT ATTRAN
    MPYK 12
    APAC
    ADDS 88
    SACL TEMP1
    LAR 1, TEMP1
    LAC *
    ADDS ATTIME
    SACL TEMP1
    CALL INSAGG            ;and put it
    B GEST,*,2             ;into list E
G6 LAR 0, TEMP4
    B G9
G10 LARP 1
    LAC *-,0,0
    SAR 1,FREE             ;extract the
    LAR 0,OLD              ;transition t
    SACL *,0,1             ;from list C
    CALL GIVEA,*+,1
    LAR 1,OLD
    LAC CONTA
    SUBS ONE
    SACL CONTA
    B G11,*,1
```

Figure 6 (cont.). Conflict management module listing.

We can expand program memory and produce an external data memory by using DSP strobe signals to handle the additional memory devices. We have not exploited this capability due to the parallel improvement in the complexity of the board.

The small size of the available memory restricts the PN's dimensions to a maximum of 90 transitions and 450 places. Conflicts cannot be handled on the TMS32010 for the same reason, though the 32010 supports all other features.

Some performance considerations. A careful investigation of the simulator's architecture permits us to identify the following distinctive parameters:

- number of parallel firing transitions (PFT),
- number of I/O places (IOP),
- number of effective conflicts (EC),
- occurrence of time-out transitions, and
- procedures associated with the transitions.

In the following section, we analyze some simple PNs to weight these parameters in their benchmark function.

Influences. The I/O places affect the simulator speed since each marking change starts a visit of the PM and P matrices. The weight of the parallel firing transitions mainly relates to the dimensions of list E.

We analyzed the PNs shown in Figures 7a and b by varying the numbers of I/O places and parallel firing transitions in the range 1-5 to evaluate their weights. We show the relevant results in Figures 8 and 9.

The results point to a linear dependence of simulator speed on the number of IOPs. This linearity derives from the proportionality between the IOP number and the visits to the matrices PM and P. Dependence on the PFT number, on the other hand, is almost quadratic, rather than linear, due to the time spent for access to list E. This list increases when the number of PFTs rises.

Figures 10 and 11 show the same results obtained on the TMS32010. The TMS32010 is 30 to 70 percent slower than the TMS32020 for two reasons. The 010 does not have a lower capability in the secondary ALU supporting indirect addressing, and a lower access speed to the part of the program memory that stores the PN.

Effective conflicts. The presence of an actual conflict clearly delays the simulator since the conflict management routine activates. This routine manages list C and extracts the conflict-winning transitions, as we stated earlier.

We analyzed the example PNs (shown in Figure 12 on page 60) by varying the EC number in the range 1-4 to estimate its influence. Figure 13 shows the results.

We can define a range of possible values, but it is impossible to actually define the processing time of the single conflicting transitions. The distribution of the time values derives from the way list C handles a transition during the filtering

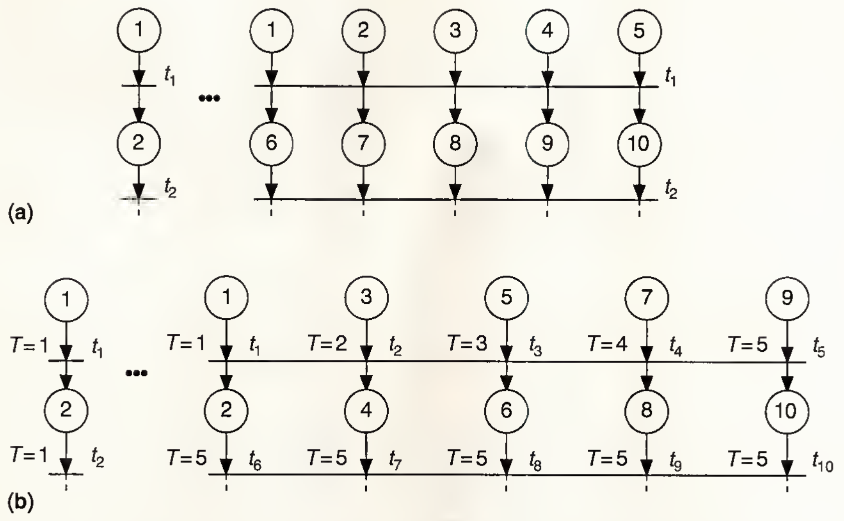


Figure 7. Some simple PNs as a benchmark for testing the I/O place (a) and parallel firing transition (b) influences on transition firing time.

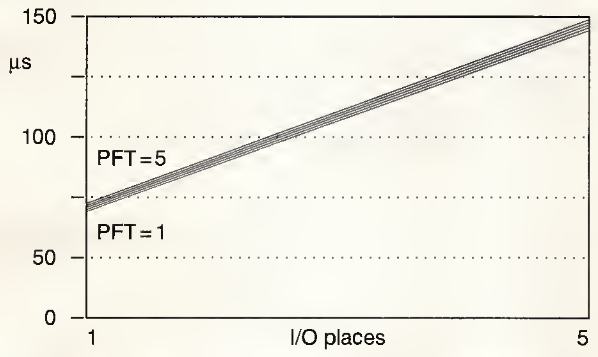


Figure 8. Dependence of the transition firing time from IOP and PFT values in TMS32020, using PFT as a parameter.

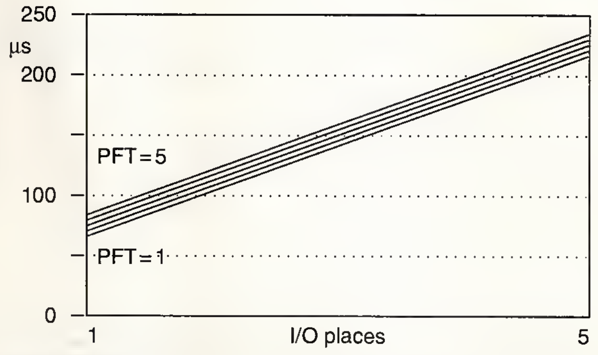


Figure 10. Dependence of the transition firing time from IOP and PFT values in TMS32010, using PFT as a parameter.

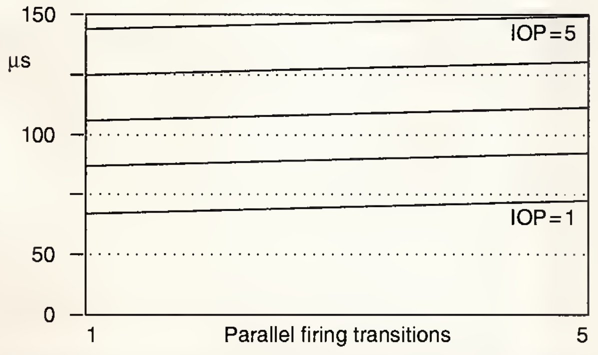


Figure 9. Dependence of the transition firing time from IOP and PFT values in TMS32020, using IOP as a parameter.

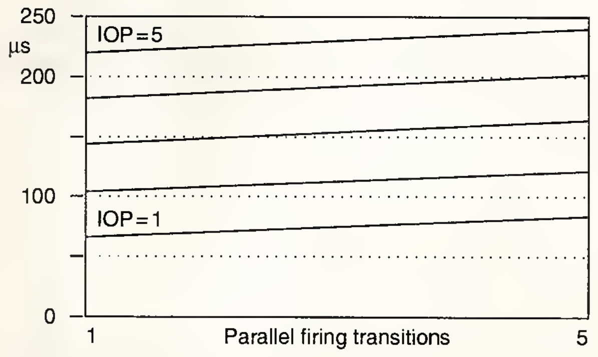


Figure 11. Dependence of the transition firing time from IOP and PFT values in TMS32010, using IOP as a parameter.

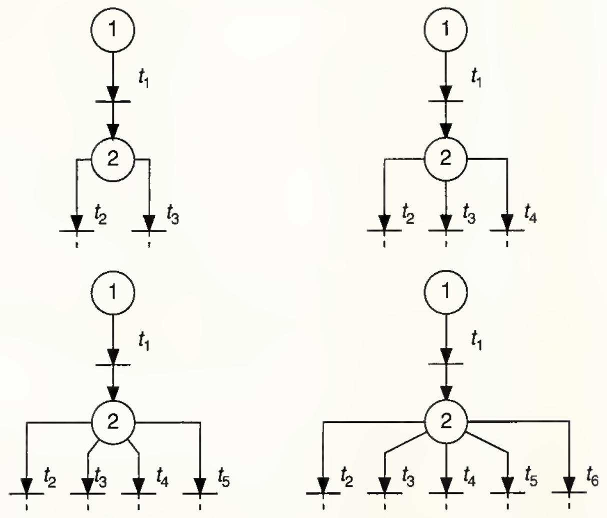


Figure 12. Some simple PNs to use as a benchmark for testing the EC influence on transition firing time.

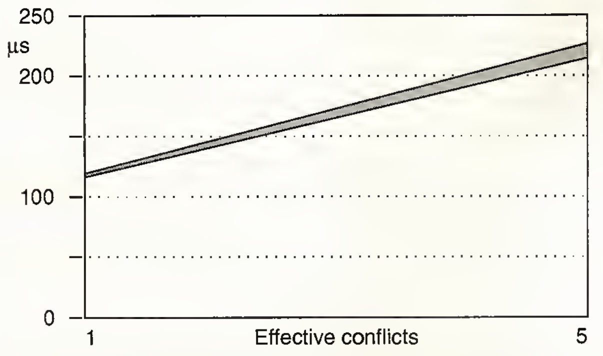


Figure 13. Dependence of the transition firing time from EC values in TMS32020.

Table 5. Values of constants appearing in formulas for time-out-transition evaluation

	Clock cycles	
Variable	TMS32020	TMS32010
A	51	162
B	112	247
C	8	9
D	20	21
E	29	29
F	11	14
G	6	4
H	11	13

process. As previously stated, the algorithm compares each transition with all the transitions in list E to check the possible presence of a common input place (actual conflict). When such a conflict does not occur, the considered transition stays in list C for a further choice.

The time required for the filtering process into list E and for inserting the subsequently activated transition(s) depends on the

- number of transitions already present in list E,
- random way of choosing the conflict-winning transition(s), and
- logical connection among the conflicting transitions.

As a consequence, the processing time of a conflicting transition varies during the tests.

In this case it is impossible to compare the overall behavior of TMS32020 and TMS32010 since conflict management does not implement on the latter.

Time-out transitions and procedures influence. We cannot adequately evaluate the time-out transitions' influence through the sampling of some simple PNs as benchmarks since the results depend on the particular PN under simulation. For this reason we chose theoretical calculation of the clock cycles spent for the ordinary time-out management.

Through a careful analysis of the algorithms produced, we derived the following formulas (see Table 5):

$$\begin{array}{ll} A & \text{if stopping\_time-out marking is discarded} \\ B + k \times C + T_{ins} & \text{if stopping\_time-out marking is realized} \end{array}$$

$$T_{ins} = \begin{cases} D & \text{if list E empty} \\ E + F \times k_1 + G \times k_2 + H \times k_3 & \text{if list E contains any transitions} \end{cases}$$

where $k \in [0,n]$ depends on the time associated with the transitions; n is the number of transitions in list E; $k_1 \in [0,n]$; and

$$k_2 = \begin{cases} 1 & \text{if } k_1 = n - 1 \\ 0 & \forall\ k_1 \neq n - 1 \end{cases}$$

$$k_3 = \begin{cases} 1 & \text{if } k_1 = n \\ 0 & \forall\ k_1 \neq n \end{cases}$$

These formulas allow users to compute the time spent in management of the ordinary stop_time-out marking. More-

continued on p. 62

Understanding petri nets

A pure PN is a suitable model for describing the dynamic evolution of a process featuring parallelism and nondeterminism. A marked pure PN is a tuple $Q = (P, T, \text{pre}, \text{post}, M)$ in which P is the place set, T is the transition set, pre means preconditions, post means postconditions, and M is the marking function.

The places represent the state variables of the described process; they can only assume discrete values. Such values are defined at any time by the application, $M: P \rightarrow \mathbf{N}$, (named marking) that globally represents the state of the process.

Transitions represent events that produce a change in the PN state. Transition firing can occur when the place marking satisfies the function, $\text{pre}: P \times T\,\mathbf{N}$. Formally, a transition $t \in T$ enables if for each $p \in P$: $\text{pre}(p, t) \leq M(p)$. A transition t is only conditioned by those places $p \in P$ for which $\text{pre}(p, t) > 0$ that are called input places of t. The enabling of a transition produces a consumption of input marks.

The state change subsequent to transition firing is described by the function, $\text{post}: P \times T\,\mathbf{N}$ (postconditions).

This function expresses the production of new marking M resulting from a transition firing. All $p \in P$ featured by $\text{post}(p, t) > 0$ are called output places of t. They are places marked by the transition firing.

In pure PNs a transition is atomic in the sense that its enabling and firing happen at the same time.

Each PN can associate with an oriented graph. Such a graph presents two different kinds of nodes: places (drawn as circles) and transitions (drawn as bars). Pre- and postcondition functions weigh and orient the arcs (only linking nodes of different kinds). Finally, black points (tokens) inside the circles indicate the place marking (see Figure A).

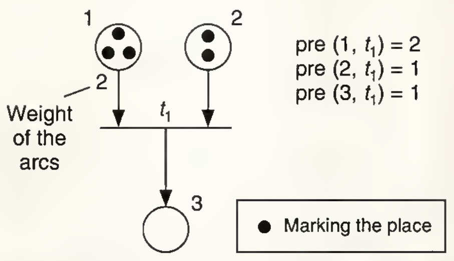


Figure A. Graphical representation of PN elements.

When an initial marking M_0 is given, all possible transition sequences describe the process behavior specified by $Q = (P, T, \text{pre}, \text{post}, M_0)$, starting from M_0.

Several transitions can fire at the same time, thus modeling parallelism.

As defined in Brahms,[10] two or more transitions $t_1, t_2 \in T$ are in structural conflict when they have at least one common input place or, more formally:

$$\exists\, p \in P \mid \text{pre}(p, t_1) \times \text{pre}(p, t_2) \neq \emptyset.$$

The structural conflict becomes actual, for a given marking M, when

$$M(p) < \text{pre}(p, t_1) + \text{pre}(p, t_2).$$

This definition reflects the actual exclusive choice between two possible firings. The conflict models the nondeterministic choice between two or more possible PN evolutions (see Figure B).

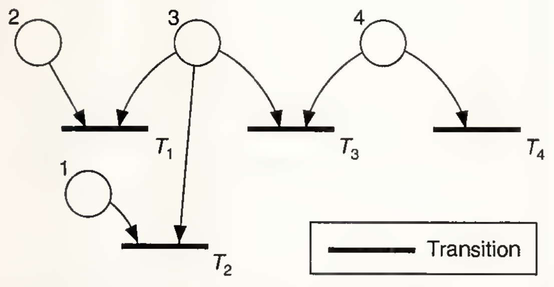


Figure B. Example of conflict occurrence and how the simulator handles it.

Inhibitor arcs. Introducing inhibitor arcs[11] improves the expressive power of PNs. They modify the firing rule as follows: a transition occurs when the input places linked to the transition by inhibitor arcs are unmarked. In a graph, every inhibitor arc ends in a white dot, as shown in Figure C.

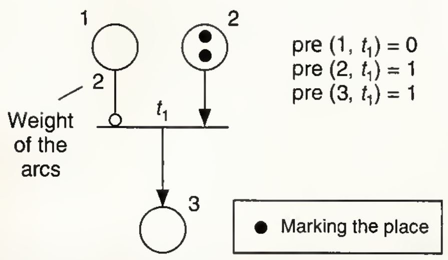


Figure C. PN transition with inhibitor arc.

Discrete time. A further improvement involves the association of a discrete time with the PN. Literature about PN temporization deals with a number of formal models.[10,12-19]

We choose to associate a discrete time with each transition by the function θ: $T \rightarrow \mathbf{N}$ (delay).

We adopted a three-phase enable/firing mechanism. When the preconditions hold, the inherent transition proceeds, consuming input tokens. The enabled transition $t \in T$ will fire after the delay time θ (t), producing a marking of the output places according to the postconditions.

Safety. At the most, safe PNs are present at one mark on each place (pre, post, and M range over {0,1}). The safety condition makes it necessary to state that an enabled transition $t \in T$ can fire when its output places are not marked. For example, this condition occurs if for each p, post(p, t) = 1, and $M(p)$ = 0.

Time-out mechanism. A useful structure we introduce in the adopted PNs is the time-out. It allows us to model conflictual behavior resolved under time conditions. We use time-out here with the classic meaning as in concurrent systems.

The time-out mechanism offers two possible mutually exclusive evolutions, according to the marking of the only two input places, called start_time-out and stop_time-out. Marking start_time-out starts the time-out mechanism at any time T_0. It implements an enabling_to_XOR_fire of two transitions, called time-out (t_1) and time-out expired (t_2). Indicating the time associated with the time-out as T, the transition t_1 fires if stop_time-out is marked within the time interval (T_0,T_0 + T).

The transition t_2 fires at time T_0 + T if time T elapses and the place stop_time-out is unmarked. Let us note that a stop_time-out marking occurring at a time-out of the (T_0,T_0 + T) range does not produce any outcome (more precisely, the mark is discarded into a drain). On the other hand, a new marking of start_time-out during the interval (0, T_0 + T) has to be considered an incorrect situation. Figure D shows the symbol of the time-out operator, while Figure E depicts the time-out description in terms of timed PNs with inhibitor arcs.

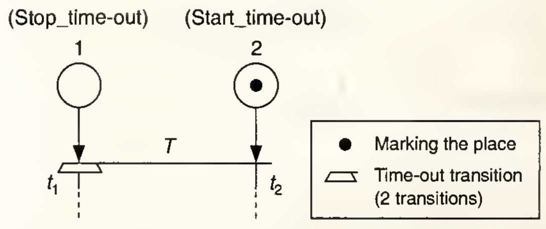


Figure D. Graphical representation of time-out mechanism.

The start_time-out's substates represented by 1′,1″, ..., 1^n allow us to show the time T elapsing into multiples of a basic time unit (e in Figure E). The transition T_{Drain} lets us discard the stop_time-out marking before the time-out mechanism starts.

Procedures

Even if the PN is a powerful formalism for representing parallelism, it lacks the common instruments of programming languages such as management of data structures, computation of algebraic expressions, and so on.

We can suitably integrate the main built-in aspect of programming languages into PNs to improve the aspect's expressiveness and handling and thus obtain a flexible, synthetic description of the system to be simulated.

We achieve this integration by associating procedures

over, they can derive a similar formula for the computation of the marking time of an ordinary nontime-out place:

$40 + 20 \times M + (14 + T_{ins}) \times M_1 + 33 \times M_2 + 3 \times T$
(TMS32020)
$73 + 35 \times M + (8 + T_{ins}) \times M_1 + 3 \times T$
(TMS32010)

where M is the number of transitions having the place as input, M_1 is the number of transitions to be inserted into list E, M_2 is the number of transitions to be inserted into list C, and T equals 1 − int (M/5).

We can compare the behavior of the two devices expressed by the above formulas by fixing, for each particular PN, the values of the parameters appearing on the devices.

We have not quantified the effect of procedures on simulation performance since their influence depends on their complexity. The delay caused by procedures must be evaluated each time it occurs.

WE ACHIEVED HIGH-PERFORMANCE LEVELS and low costs as a result of our implementation of the simulation tool.

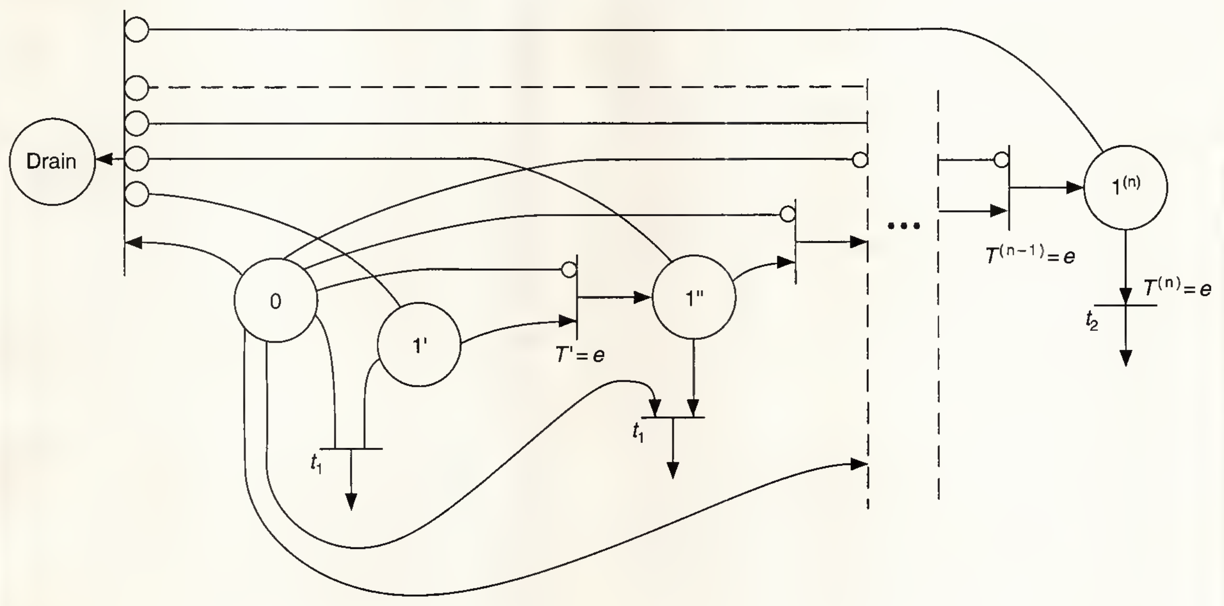


Figure E. Time-out model by timed transitions and inhibitor arcs (T = n × e).

with transitions. We can associate a procedure in a high-level language or in the assembler to each $t \in T$ transition. A function Proc resumes the correspondence between a transition t and a procedure identifier Id ∈ ProcId,

Proc: $T \rightarrow$ ProcId

The process described by the procedure Proc(t) starts when the transition fires. The procedure can be considered atomic in the sense that all the actions it performs must be completed before processing any other transition.

When an enabled transition has one associated procedure, the system links with a procedural environment at the start of the procedure. The process control switches to the new environment where the procedure executes. The procedure can both set the control variables and change the present PN configuration.

Then the process control shifts back to the PN environment modified if the procedure provides for this after procedure execution.

The PN class outlined compactly represents a wide range of processes by restricting the model size and the relevant simulation times. A qualified, high-efficiency simulator can attain a nonredundant algorithm—agile in manipulating the data structure—implementing it with machine language on fast hardware such as the DSP board.

The use of a parallel hardware architecture to implement parallel evolution of the modeled processes, instead of simulating them on a single, though fast, processor such as the DSP used here could further improve the performance parameters considered.

References

1. P. Azema et al., "Specification and Verification of Distributed Systems Using Prolog Interpreted Petri Nets," *Proc. Seventh Int'l Conf. Software Eng.*, 1984.
2. M. Diaz, "Petri Nets Based Models in the Specification and Verification of Protocols," *Lecture Notes in Computer Science*, Springer-Verlag, Bad Hormel, Germany, No. 255, 1986, pp. 135-170.
3. M.K. Molloy, "Performance Analysis Using Stochastic Petri Nets," *IEEE Trans. Computer*, Vol. C-31, No. 9, 1982, pp. 913-917.

4. M.A. Marsan, G. Balbo, and G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," *ACM Trans. Computer Systems*, Vol. 2, No. 2, May 1984, pp. 93-122.
5. C.V. Ramamoorthy and G.S. Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," *IEEE Trans. Software Eng.*, Vol. SE-6, Sept. 1980, pp. 440-449.
6. J. Sifakis, "Petri Nets for Performance Evaluation Measuring, Modeling, and Evaluating Computer Systems," *Proc. Third Int'l Symp.*, Int'l Federation for Information Processing Working Group 7.3, Geneva, 1977, pp. 75-93.
7. T. Murata, et al., "A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and its Applications in Factory Automation," *IEEE Trans. Industrial Electronics*, Vol. IE-33, No. 1, 1986, pp. 1-8.
8. *TMS32010 User's Guide*, Texas Instruments, Houston, TX, 1985.
9. *TMS32020 User's Guide*, Texas Instruments, Houston, TX, 1985.
10. G.W. Brahms, "Resaux de Petri: theorie et pratique, Tome 2—Modelisation et applications," ("Petri Networks: Theory and Practice, Vol. 2, Modelization and Application"), ESI, Paris, 1983.
11. M. Hack, "Petri Nets Languages," Computing Structures Gr. Memo 124, Project HAC, Massachusetts Institute of Technology, Cambridge, Mass., June 1975.
12. W.M. Zuberek, "Timed Petri Nets and Preliminary Performance Evaluation," *Proc. Seventh Ann. Symp. Computer Architecture*, 1980, pp. 88-96.
13. M.A. Holliday and M.K. Vernon, "A Generalized Timed Petri Nets Model for Performance Analysis," *Proc. Int'l Workshop Timed Petri Nets*, IEEE Computer Society Press, July 1985.
14. W.M. Zuberek, "M-Timed Petri Nets, Priorities, and Performance Evaluation of Systems," *Advances in Petri Nets 1985, Lecture Notes in Computer Science*, Springer-Verlag, No. 222, 1986.
15. M. Menasche and B. Berthomieu, "Timed Petri Nets for Analyzing and Verifying Time Dependent Communication Protocols," *Protocol Specification, Testing and Verification*, IFIP III, 1983.
16. A. Faro, O. Mirabella, and C. Nigro, "Computer Network Analysis by Using a Generalized PN Simulator," *Proc. Mathematics and Computer in Simulation XXVI*, North Holland, Amsterdam, 1984, pp. 401-411.
17. A. Faro, O. Mirabella, and C. Nigro, "Performance Evaluation of the Standard PROWAY Network for Process Control," *IEEE Trans. Industrial Electronics*, Feb. 1986, pp. 11-20.
18. A. Di Stefano and O. Mirabella, "Throughput and Reliability Improvement in a Multi Packet Multiring Lan," *Proc. 16th Ann. Pittsburgh Modeling and Simulation Conf.*, IEEE CS Press, Los Alamitos, Calif., Apr. 1985, pp. 781-786.
19. C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets," PhD thesis, MIT, Sept. 1973.

Antonella Di Stefano is a researcher at the Institute of Informatics and Telecommunications at the University of Catania, Italy. Her research interests include formal description techniques, petri-net-based automated simulation tools, and performance evaluation of parallel architectures. She received a graduate degree in electrical engineering from the university.

Orazio Mirabella is associate professor of computer science at the University of Catania. His research interests include DSP-based system design, performance evaluation of parallel architectures and local area network protocols, and AI-based techniques. He received a graduate degree in physics from the university.

Fabio Presente is a researcher at the Center for Advanced Studies for Information Technologies, a research institute for the development of advanced techniques. His research interests include DSP-based systems design and techniques for natural language understanding. He received a graduate degree in electrical engineering from the university.

Address all questions concerning this article to the authors at the Istituto di Informatica e Telecomunicazioni, Facolta di Ingegneria, V.le A. Doria 6, 95125, Catania, Italy.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 156 Medium 157 High 158

continued from p. 27

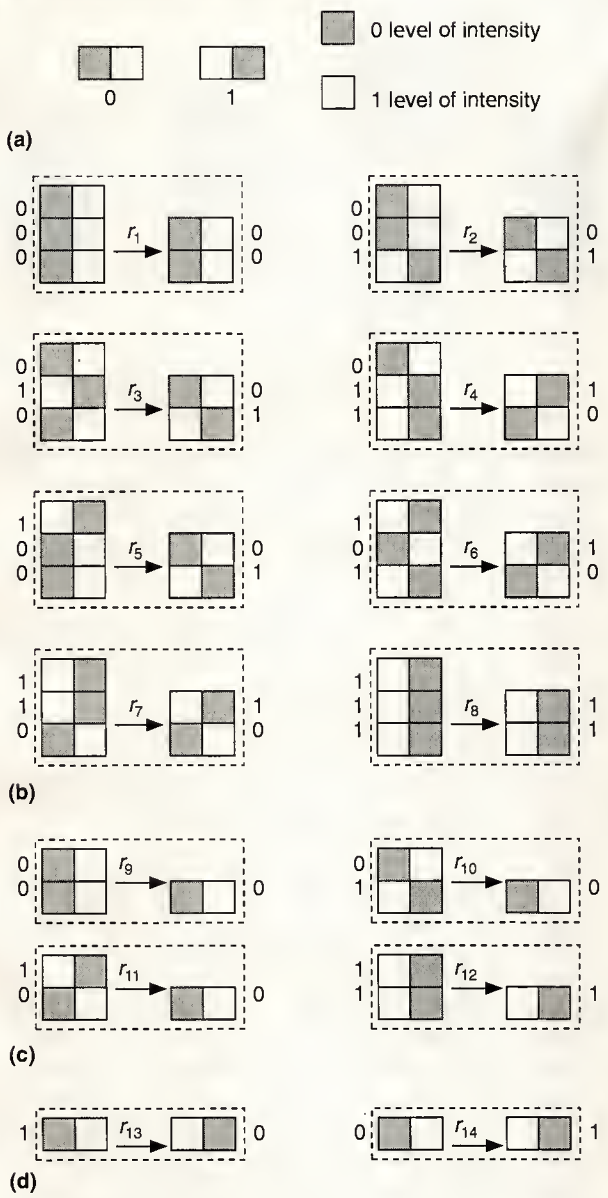


Figure 3. Light-intensity encoding of the binary values 0 and 1 (a); optical SSL rules for primitive operators: the full Add (b), the logical And (c), and the logical Not.

pixels, dark and bright, and the logic value 1 by the inverse pattern, bright and dark, as shown in Figure 3a. The dark and bright pixels represent increasing levels of light intensity. In this dual-rail coding scheme, the intensity of the bright pixel and its position represents a logic value, which has some implementation advantages.[43] We refer to the optical encoding of the binary values 0 and 1 as the fundamental patterns.

We next implement the fundamental operators as SSL rules, specifying how to manipulate information represented by optical patterns. These optical patterns are combinations of the fundamental patterns.

We derive the SSL rules from the truth-table specifications of each operator and the fundamental patterns shown in Figure 3a. The input combinations of the truth tables represent the search patterns of the SSL rules, while the table entries represent the replacement patterns. The P-Add operator truth table manipulates three bits, which gives rise to the eight combinations shown in Figure 3b. If we place the bit symbols on top of each other, we produce eight SSL rules. Note that a separate input plane provides each bit. These bits have the same coordinates *i, j* in each plane.

The 2D 3-shuffle function described earlier groups bits of the same coordinates. Similarly, the logical P-And and P-Not give rise to four and then two SSL rules, as shown in Figure 3c,d. Thus, we need a total of 14 SSL rules to implement the three fundamental operators.

Implementation of SSL. We briefly illustrate the implementation of one SSL rule (r_3 in Figure 3b) using an additive logic implementation method[43,55] to assist the conceptual understanding of this technique. The required optics have two parts: pattern recognition followed by pattern replacement. The recognition phase locates the presence of the search pattern in the input image, while the substitution phase uses this information to substitute the replacement pattern. The recognition optics applies a thresholding operation to a composite of shifted replicas of the input image. See Figure 4.

Using dual-rail coding (Figure 3a) and assuming dark-pixel recognition, the pattern-recognition optics replicate the input image as many times as there are dark pixels in the search pattern (Figure 4b). Then each replica shifts horizontally and/or vertically by an amount that brings a corresponding dark pixel to a designated reference pixel (Figure 4c). The shifted replicas are then superimposed optically (Figure 4d), and an optical Nor-gate array inverts the resulting image (Figure 4e). We mask the output of the Nor-gate array to eliminate erroneous overlapping patterns.

The masked image constitutes the recognition plane, since each bright pixel in it indicates the presence and location of the search pattern (Figure 4f). This image enters the pattern replacement phase (Figure 5), which replicates the recognition image for each bright pixel in the replacement pattern (Figure 5b). Then each replica shifts by an amount corresponding to the position of the bright pixel associated with it in the substitution pattern (Figure 5c). These shifted replicas then form the final image, shown in Figure 5d, through optical superimposition. We omitted optical hardware for image

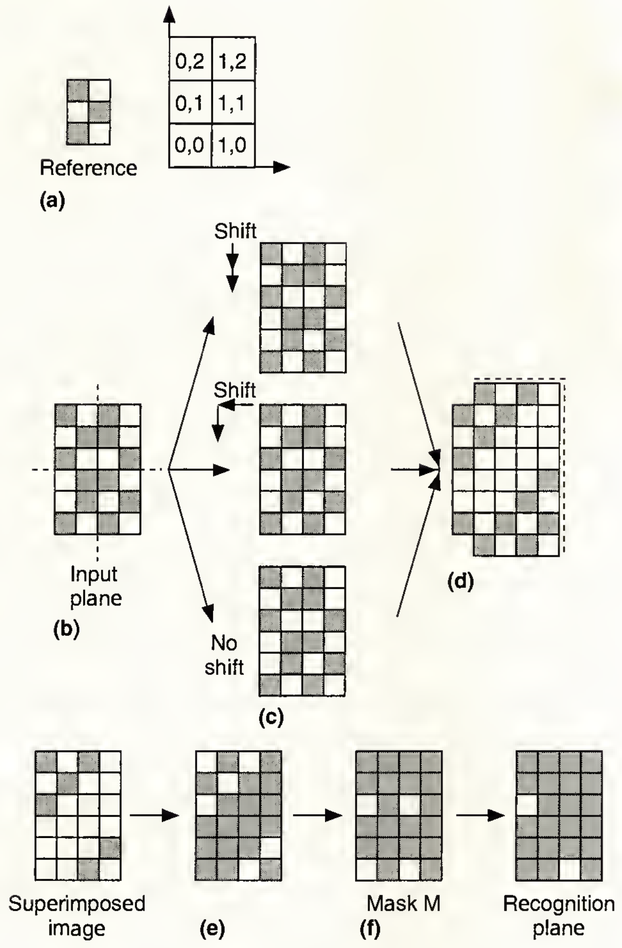


Figure 4. Processing steps needed to implement the recognition phase of SSL using additive logic: search pattern (a), replication (b), shift (c), superimposition (d), inversion (e), and masking (f).

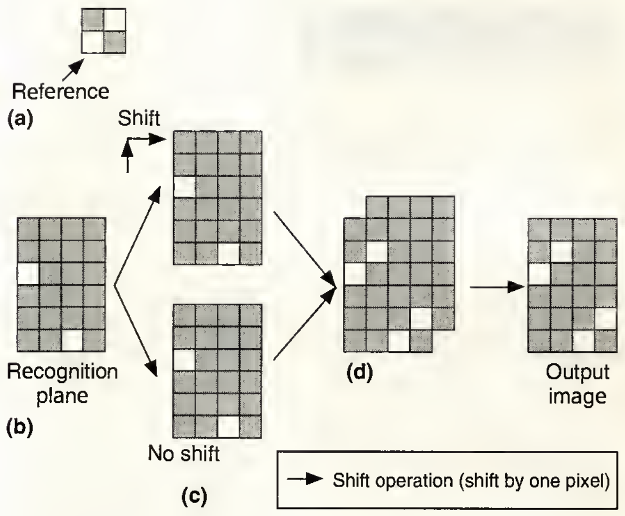


Figure 5. Optical processing steps needed to implement the substitution phase of SSL using additive logic: replacement pattern (a), replication (b), shift (c), and superimposition (d).

replication, shift, combination, and masking from Figure 5 for clarity. Brenner et al.[43] provides a detailed description of this particular method, including the optical setup.

Implementation of the processor array. Each of the three fundamental operators comprises several SSL rules that need to be fired simultaneously. To do so, we replicate the output of the input combiner a number of times equating the number of SSL rules to be activated at a given stage of computation.[43]

For example, to perform the P-Add operator, we need to replicate the input plane eight times corresponding to the eight SSL rules associated with the P-Add operator. Passive optical components such as beam splitters, or holographic elements can replicate the input. However, we would need a binary treelike replication scheme to equalize the optical path for each copy.

Each copy moves to one of the eight SSL rules r_1 to r_8 of Figure 3b. After the necessary substitutions, we optically superimpose the outputs of every active SSL rule to form the processed result. The optical superimposition represents a logical Or of light patterns in which a bright pixel overwrites a dark pixel. Thus we can implement the processor array with three modules, namely, an Add, an And, and a Not.

Each module comprises the SSL rules of the corresponding operator, as illustrated in Figure 6. A dynamic beam-steering element (an acousto-optic or electro-optic deflector along with some mirrors) under program control deflects the input plane to the desired module. Within each module, a static beam-steering element directs the processed output to the output router as shown in Figure 7. Recently, I[9,53] introduced an alternative dynamic method to implement the processor array that does not require a dynamic steering device.

Implementation of data-routing functions. The input combiner and output router assume only data movement functions; they do not require data processing. The input combiner assumes the three functions already described. Transmitting a bit plane to the processor array does not involve permutation of the data, and therefore we can use any imaging system.

The 2D perfect shuffle and 3-shuffle functions permute the row position of the input data. The literature proposes a wide variety of methods for realizing these functions.[56-60] These

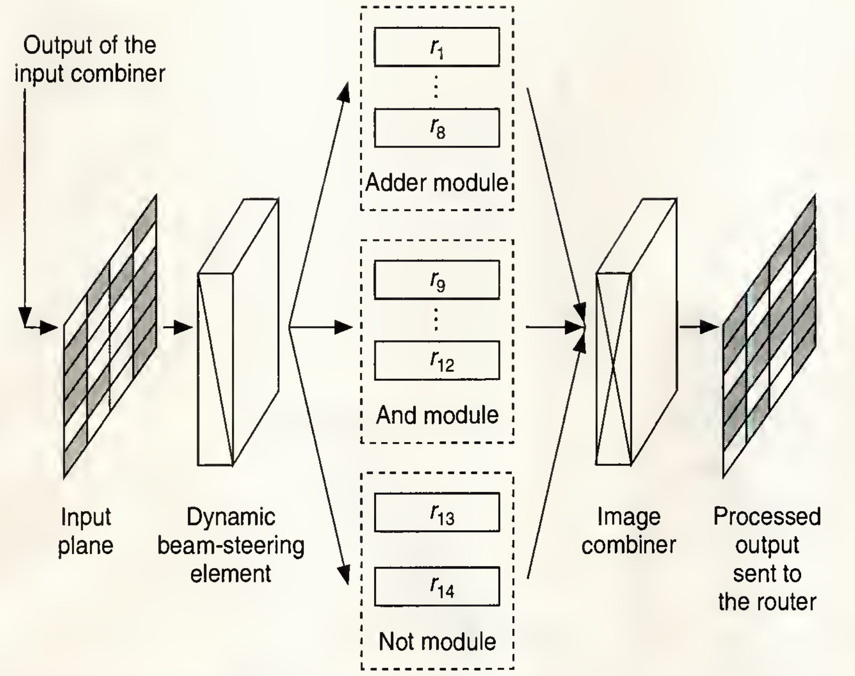


Figure 6. Logical structure of the optical processor array.

masking. The same principle can be extended to implement the 2D perfect shuffle and 3-shuffle functions described here.

Figure 8 on the next page illustrates an optical implementation of the 2D perfect shuffle, extended from Lohmann et al.[56] Given two bit planes, we implement the 2D perfect shuffle by first magnifying each bit plane to the total size of the two input planes and then achieve the 2D permutations by appropriate masking. Polarization-based devices controlling the flow of data control this setup. Similar considerations take place for the 3-shuffle function.

Polarization control devices can also implement the data movement functions assumed by the output router (feedback, routing data to memory, and shifting). We can use polarizing beam splitters and halfwave plates to control the pathways of the light beam. Two sources[61,62] suggest using birefringent prisms and acousto-optic cells to perform uniform shifts of a bit plane. Drabik and Lee[28] also proposed real-time holograms (discussed later) in photorefractive media to accomplish space-invariant shifts. The output-input optical feedback is one-to-one mapping that does not require permutation of the data; furthermore it does not need to be reconfigurable, which renders its optical implementation very simple. In fact an imaging system with some control (polarization-based devices) can implement the optical feedback.

schemes can be made controllable (whether to shuffle) using electro-optical and polarization-based devices.[61]

Lohmann et al.[56] proposed an optical setup for performing perfect shuffle permutations of a one-dimensional column of optical data using geometrical optical devices. These devices use the inherent speed of light and dissipate very little power. The basic principle is to divide the input into two upper and lower portions, magnify the two halves to the original size of the input, and then obtain the shuffled output by appropriate

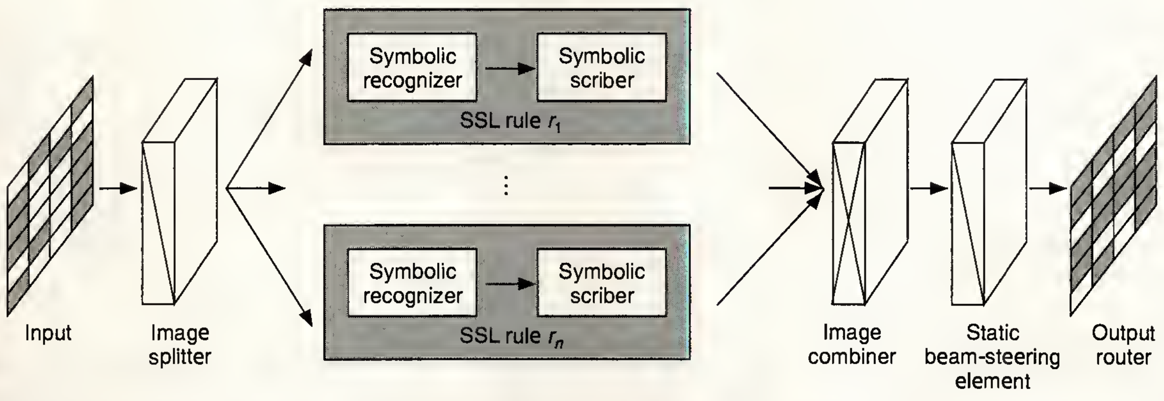


Figure 7. The logical implementation of each functional module in Figure 6.

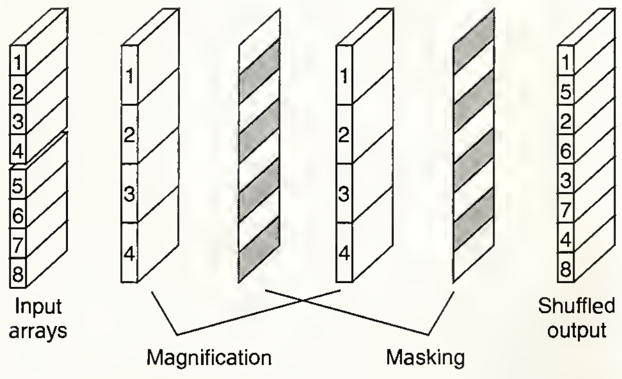


Figure 8. Optical implementation of the 2D perfect shuffle permutations. The numbers in the boxes represent the row positions within the plane. We achieve the 2D shuffle function by interlacing the upper and lower halves of the input.

Memory organization and control. To maintain the 2D processing throughout the system, the data memory must be addressable in bit planes. For single-plane storage, such as the input and output planes, and temporary buffers, we can use spatial light modulator (SLM) technology[63,64] and bistable optical latches.[15,16,21,22,65] SLMs are real-time optical active devices capable of spatially or temporally modifying some characteristic (polarization, phase, amplitude, intensity) of an optical signal beam. SLMs have a broad range of applications in optical information processing, including image amplification, inversion, incoherent-to-coherent conversion, analog multiplication, wavelength conversion, and short-term storage. However, SLMs would not be sufficient to build a data memory unit capable of holding a large number of bit planes.

Volume holograms, with the capability to store information in three dimensions, show the potential for a dramatic increase in optical storage density (10^{11} to 10^{12} bits). By recording stacked holograms in photorefractive crystals, we can achieve high storage density in random-access optical memory.[66-69] Bragg angular selectivity allows superpositions of holographic planes by slightly changing the angle between the recording beams.[62] Erasing such recorded holograms usually takes place by uniformly illuminating the storage crystal or by heating.

A holographic memory unit would be a superposition of volume holograms, one for each bit plane. Figure 9a schematically indicates how information moves into the optical memory in plane format. The bit planes from the output router form the data to be stored. Each bit plane is encoded in a volume hologram in which spatial frequencies are characteristic of a unique reference beam as well as the light distribution on the bit plane. The controller generates the memory address indicating where the data should be stored. The beam

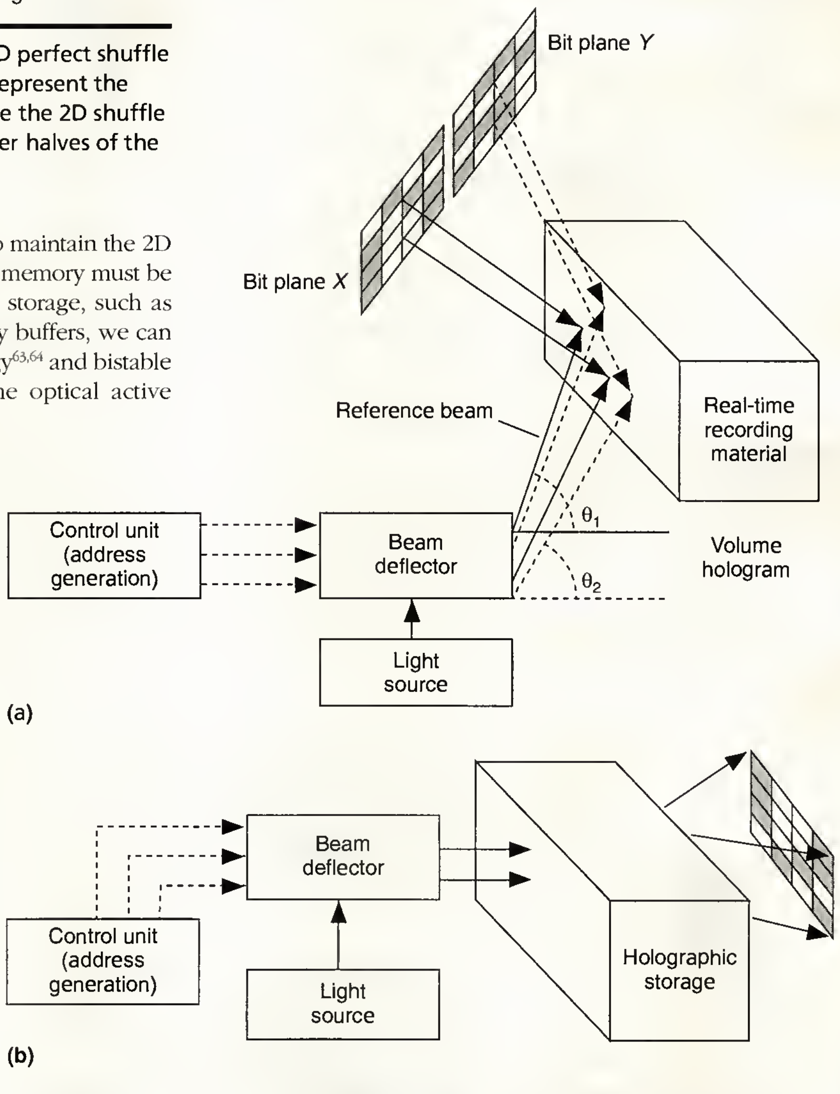


Figure 9. Real-time storage in a volume holographic medium (a) and retrieval from a volume holographic memory (b) of a light-encoded bit plane. The angular positions θ_1 and θ_2 correspond to the physical addresses of bit planes *X* and *Y*.

deflector device deflects the light by an amount proportional to the address generated by the controller. The interference of the image plane and the reference beam is recorded in the volume hologram.

Figure 9b depicts data retrieval from the holographic storage. The controller generates the address of the bit plane to be read and sends it to the beam deflector. This device in turn illuminates the volume holographic unit by a reference beam with an angular direction corresponding to the address generated by the controller. It should be noted that this angular position is the same as the one used to record the bit plane in the volume hologram.

Although this technology is very promising for real-time optical storage, it is far from being perfected for use in main memory applications. Several severe problems have to be solved before it becomes practical. These include cross talk between multiple holograms stored in the medium, low-diffraction efficiency of multiple volume holograms, fast selective writing and erasure of data, and data volatility. Nevertheless, Redfield and Hesselink[70] report making major efforts to overcome these limitations.

Another way of implementing the data memory would be to extend the optical disk technology. A 15-cm optical disk may have as many as 40,000 tracks and contain up to 10 Gbits of storage,[71,72] which corresponds to one thousand 1,000 × 1,000 bit planes. Recently, research efforts have concentrated on read/write optical disks based in magneto-optic combinations.[8,71]

A thin layer of vertically oriented material (a rare-earth, transition-metal alloy such as gadolinium and terbium) that is sandwiched between a transparent polycarbonate protective coating and a reflective substrate constitutes the recording medium. Initially, the orientation of all data bits is the same (corresponding to logic level 0). To write a bit on the disk, we use a high-intensity laser beam to heat the spot on the disk, dropping its coercivity with increasing temperature. This drop makes it possible to magnetize the heated spot easily with a weak magnetic field when applied.

Readout, based on the Kerr effect, results from using a lower beam to illuminate the location of each bit on the disk individually. Based on the reflected or transmitted intensity detected, the bit is decoded as logical 0 or 1. For writing, we can use the same optical setup for writing data on the disk. The key feature of magneto-optical disks is their use as parallel-access (read/write) memories. By illuminating a large portion of the disk during a write cycle or a read cycle, we can access several bits of information for writing or reading at once.[8,73]

Conceivably, an optical bit-plane addressable memory unit based on the transmissive optical disk technology could be built as shown in Figure 10. A controller sends the address of the bit plane to be read to the disk controller, which translates it into a mechanical motion of the disk. Meanwhile a different signal simultaneously moves to a laser deflection device that shines a light beam on the exact location of the data. Thus an entire bit plane can be read at once. According to the Faraday effect, the polarization of a laser beam passing through a magnetized medium experiences a rotation determined by the direction of the magnetic field of the material. Hence, magneto-optical disks can operate in the transmissive mode, provided that we take appropriate measures to eliminate diffraction effects.

Although the optical disk technology is commercially available for secondary storage (the Next computer uses a 256-Mbyte, read/write magneto-optical disk as a secondary storage device), it is limited. The main limitations of this technology for real-time main memory storage required for optical computing are the high access time and the relatively immature optical read/write schemes and devices (typical

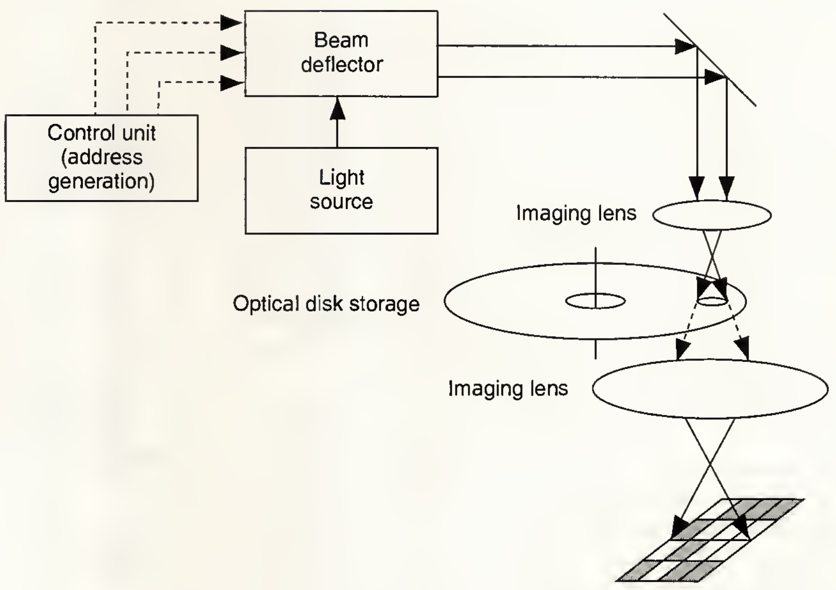


Figure 10. Using an optical disk as a real-time read/write memory device. We read the address of the bit plane from memory and access the desired bit plane of data at once.

access time for read/write optical disk is 100 ms).[8,62] The high access time results from the rotational latency or the delay incurred while waiting for the desired data to rotate to the proper location.

A potential solution to the mechanical motion problem may be the development of 2D optical beam deflectors that can provide tens of thousands of beam positions and very fast deflection time. In this manner, the disk will become stationary, and disk access will take place through the optical beam-deflector device. Henshaw and Todtenkopf[74] report several techniques under consideration: wavefront tilt, phased array, polarization modulation, interferometric switching, and photorefractive beam steering.

Another alternative for implementing the optical bit-plane addressable memory is the use of the two-photon-based, 3D optical memories.[75] Researchers claim that the two-photon effect provides a means of storing data into separate bit locations throughout the volume without affecting the neighboring bit locations. Thus the effect provides the highest storage density and the largest bandwidth of any existing storage device. In addition, this process permits a higher accessing speed than that found in volume holographic storage. These claims have been demonstrated.

2D computing substructures. The optical architecture exploits spatial parallelism at the hardware level, which enables it to process an entire data plane at once. This capability is opposed to task (or function) parallelism in which the data plane is decomposed into subplanes that are processed sequentially in a pipelined fashion.[76]

To enforce this capability at the algorithm design level, we view the design and mapping process as a hierarchical structure, as shown in Figure 11. At the highest level of the hierarchy is the application we wish to solve (signal and image processing, vision, radar). The next level identifies the various algorithms we can use to compute these applications. This level includes matrix algebra, numerical transforms, and solutions of partial differential equations among others. A further analysis of these algorithms reveals that they share a common set of high-level operations, which we call computing substructures. These substructures can in turn be decomposed into a set of fundamental operators such as the P-Add, the logical P-And, and P-Not.

The rationale behind this mapping technique is that most of the data-parallel algorithms share common attributes such as regularity, localized and intensive computations, recursiveness, and matrix operations. So the mapping process starts by identifying a set of substructures that captures most of these features. We then must efficiently map these substructures onto the architecture and build parallel algorithms upon them. This makes the mapping process more systematic and hence efficient. In this article we concentrate on a representative set of these computing substructures to show the methodology.

We will denote data transfer by A (B or C) $\leftarrow X_k$ by which we mean the transfer of bit plane X_k to the input plane A (B or C). Similarly, the expression $X \leftarrow Y$ denotes the transfer of bit plane Y to bit plane X. This step involves loading Y from memory, going through the processor array and the output router without any effect, and storing it in X.

We use $C \leftarrow 0$ plane to clear all the entries of input plane C. Similarly, $C \leftarrow 1$ plane indicates the setting of all the entries of input plane C to 1, and $P \leftarrow 0$ plane denotes a transfer of a zero-bit plane to memory location P. Loop constructs such as $k := a$ to $\log_2 n$ and indices such as a and $\log_2 n$, and parameter calculations should be interpreted as control instructions that the control unit executes.

2D addition/subtraction. This substructure refers to the addition (or subtraction) of corresponding elements of two $n \times n$ data planes $\mathbf{X}$ and $\mathbf{Y}$ of integers. The result is a data plane $\mathbf{S} = \{\mathbf{s}_{ij}\}$, whose elements $\mathbf{s}_{ij} = \mathbf{x}_{ij} \pm \mathbf{y}_{ij}$ for $i, j = 1, \ldots, n$. This step is similar to conventional matrix addition (subtraction). Let $\mathbf{X}$ be an $n \times n$ q-bit planes, $X_{q-1}, X_{q-2}, \ldots, X_0$. Here q is the precision of the operands, X_0 being the least significant and X_{q-1} being the most significant bit planes respectively. Similar considerations take place for the data plane $\mathbf{Y}$.

The 2D addition substructure adds the corresponding elements of the data planes bit serially, starting from the least

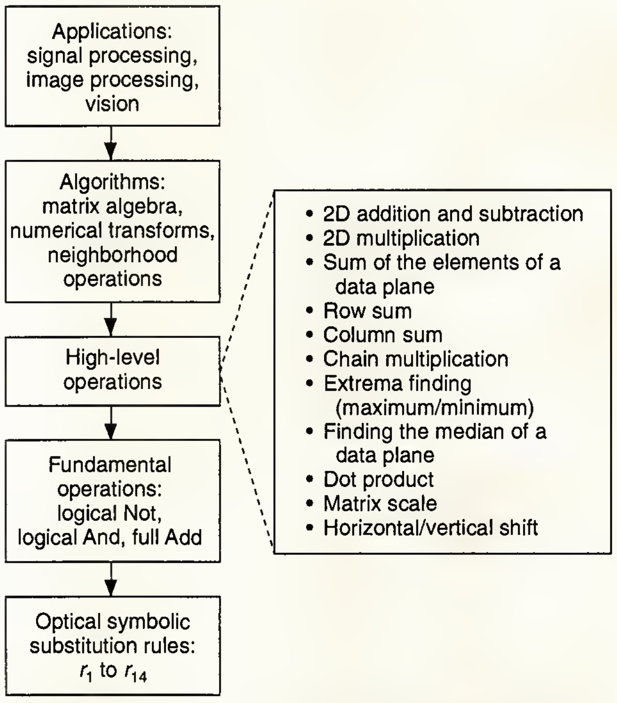

Applications: signal processing, image processing, vision
Algorithms: matrix algebra, numerical transforms, neighborhood operations
High-level operations
Fundamental operations: logical Not, logical And, full Add
Optical symbolic substitution rules: r_1 to r_{14}
• 2D addition and subtraction
• 2D multiplication
• Sum of the elements of a data plane
• Row sum
• Column sum
• Chain multiplication
• Extrema finding (maximum/minimum)
• Finding the median of a data plane
• Dot product
• Matrix scale
• Horizontal/vertical shift

Figure 11. A top-down approach to mapping algorithms onto the bit-plane architecture.

significant bit planes. The substructure starts by initializing the C plane to zero and loading bit planes X_0, Y_0 into the A plane and the B plane respectively. The input combiner performs the 3-shuffle function of the three input planes. The processor array applies the P-Add SSL rules simultaneously to the resulting image. Thus the addition proceeds on all the operand pairs in parallel.

The sum bits are extracted from the output plane and stored in memory location S_0, and the carry bits are extracted and fed back to the C plane for the next iteration. Meanwhile the memory unit loads bit planes X_1 and Y_1 in the A plane and B plane respectively. The whole process continues until X_{q-1} and Y_{q-1} are added, and the sum $S_0, S_1, \ldots, S_q$ is stored as stacks of bit planes in the memory.

```
Procedure 2D Addition(X,Y)
  begin
      $C \leftarrow 0$ plane ;
      /* initial carry is zero */
      for $k := 0$ to $q - 1$ do
          /* begin $k$ loop */
              $A \leftarrow X_k$;
              /* load contents of $X_k$ into $A$ plane */
              $B \leftarrow Y_k$;
              /* load contents of $Y_k$ into $B$ plane */
              $S_k, C \leftarrow$ P-Add($A,B,C$)
              /* Perform the full addition of input planes
                     $A, B, C$*/
      endfor              /* end $k$ loop */
      $S_q \leftarrow C$;
      /* transfer the last carry to the most significant bit
                     plane $S_q$ of S */
  end 2D Addition
```

The notation S_k, $C_{out} \leftarrow$ P-Add(A,B,C) in this procedure designates the addition of bit planes A and B together with the previous carry C_{in}. The sum bit plane moves to storage in S_k, and the resulting carry bit plane C_{out} returns to input plane C. We add two q-bit planes in q iterations, regardless of the number of operands to be added.

Representation of numbers in two's-complement form allows 2D subtraction by adding few additional steps to the 2D addition procedure. To subtract two data planes **X,Y**, we first form the two's complement of the subtrahend **Y** and then add it to **X** using the 2D addition procedure. We obtain the two's complement of data plane **Y** by first negating all the bit planes of **Y** ($Y_i \leftarrow Y'_i$ for $i = 0, \ldots, q-1$ using the P-Not operator). We then add it to a data plane whose least significant bit plane is a 1 plane; the remaining $q - 1$ bit planes are all 0 planes.

2D multiplication. This operation refers to the multiplication of corresponding elements of two data planes. Let **X** and **Y** be $n \times n$ q-bit planes. The product **P** forms as $2q$-bit planes $\mathbf{P} = P_{2q-1}\, P_{2q-2}, \ldots, P_0$, where $\mathbf{P}_{ij} = \mathbf{x}_{ij} \times \mathbf{y}_{ij}$. As an example, let q be equal to 3 (we assume the same precision for both data planes to simplify the example). With $\mathbf{X} = X_2X_1X_0$ and $\mathbf{Y} = Y_2Y_1Y_0$, the resulting product then becomes $\mathbf{P} = P_5P_4P_3P_2P_1P_0$. The multiplication process starts by clearing the product bit planes to zero:

$$P_k \leftarrow 0 \text{ plane for } k = 0, \ldots, 5. \qquad (3)$$

This step represents the initial partial product $\mathbf{P}^0$ (the superscript 0 indicates the initial partial product). Next, we calculate the first partial product $\mathbf{P}^1$:

$$\begin{array}{ll}
P_0^1 & \leftarrow \text{P-And}(X_0, Y_0) \\
P_1^1 & \leftarrow \text{P-And}(X_1, Y_0) \\
P_2^1 & \leftarrow \text{P-And}(X_2, Y_0) \\
P_3^1 & \leftarrow P_3^0 \\
P_4^1 & \leftarrow P_4^0 \\
P_5^1 & \leftarrow P_5^0
\end{array}$$

The notation P_j^i ($j = 0, \ldots, 5$) means the jth bit plane of the ith partial product. The second partial product $\mathbf{P}^2$ is generated from $\mathbf{P}^1$ in the following manner:

$$\begin{array}{ll}
P_0^2 & \leftarrow P_0^1 \\
C & \leftarrow 0 \text{ plane} \\
T & \leftarrow \text{P-And } (X_0, Y_1) \\
P_1^2, C & \leftarrow \text{P-Add } (P_1^1, T, C) \\
T & \leftarrow \text{P-And } (X_1, Y_1) \\
P_2^2, C & \leftarrow \text{P-Add } (P_2^1, T, C) \\
T & \leftarrow \text{P-And } (X_2, Y_1) \\
P_3^2, P_4^2 & \leftarrow \text{P-Add}(P_3^1, T, C) \\
P_5^2 & \leftarrow P_5^1
\end{array}$$

The variable T here is a temporary bit plane. We obtain the final product after three iterations $\mathbf{P} = \mathbf{P}^3$. This product is produced as:

$$\begin{array}{ll}
P_0^3 & \leftarrow P_0^2 \\
P_1^3 & \leftarrow P_1^2 \\
C & \leftarrow 0 \text{ plane} \\
T & \leftarrow \text{P-And}(X_0, Y_2) \\
P_2^3, C & \leftarrow \text{P-Add}(P_2^2, T, C) \\
T & \leftarrow \text{P-And}(X_1, Y_2) \\
P_3^3, C & \leftarrow \text{P-Add}(P_3^2, T, C) \\
T & \leftarrow \text{P-And}(X_2, Y_2) \\
P_4^3, P_5^3 & \leftarrow \text{P-Add}(P_4^2, T, C)
\end{array}$$

Note that, unlike the conventional shift and add multiplication algorithm, we did not need to shift the previous partial product to generate the current one. Instead, we start the addition at the bit plane corresponding to the amount of shift required. The complete procedure is as follows:

```
Procedure 2D Multiplication(X,Y)
  begin
  for $k$ := 0 to $2q - 1$ do
  /* this loop clears the bit planes of the product to 0 */
     $P_k \leftarrow$ 0 plane;
  for $l$ := 0 to $q - 1$ do
  /* this loop generates the successive partial products */
        $C \leftarrow$ 0 plane;
        for $m$ := 0 to $q - 1$ do
           $A \leftarrow X_m$;
           $B \leftarrow Y_l$;
           $B \leftarrow$ P-And($A,B$);
           $A \leftarrow P_{m+l}$;
           $P_{m+l}, C \leftarrow$ P-Add ($A,B,C$);
        endfor;  /* end of $m$ loop */
        $P_{q+l} \leftarrow C$;
  endfor;          /* end of $l$ loop */
end 2D Multiplication
```

It takes q^2 full additions and q^2 logical And operations to generate the final product **P**. Therefore, the time complexity of the 2D multiplication is $0(q^2)$, independent of the number of pairs to be multiplied.

2D data-shifting operations. We define two operations for shifting a data plane by a variable number of pixels in either direction. The logical shift involves columns (or rows) of 0s that enter from the opposite side of the shift direction. Given a data plane **P** of q-bit planes P_{q-1} P_{q-2}, ..., P_0, we define a horizontal shift operation, denoted by $\mathbf{H}_\alpha$ **(P)**, to be the data plane **P** shifted in the X axis by α columns (+ α, for positive shift, and – α for negative shift). See Figure 12a.

The amount of shift is sequentially applied to every bit plane P_i of the data plane **P**. The shifted plane can either be stored in itself or in a different data plane in memory. For the latter case, we introduce the notation $\mathbf{X} \leftarrow \mathbf{H}_\alpha(\mathbf{P})$, by which we mean that the shifted plane **P** is stored in plane **X**. Similarly, we define two other operations, denoted by $\mathbf{V}_\alpha(\mathbf{P})$ and $\mathbf{X} \leftarrow \mathbf{V}_\alpha(\mathbf{P})$ for vertical shifting. An illustration of vertical shift apperars in Figure 12b.

We now have an optical register-transfer language, comprising the 2D operations just described, with which we can describe parallel algorithms without referring to the machine hardware. Hereafter, the following shorthand notations **2D add(X, Y), 2D sub(X, Y), 2D multiply(X, Y),** $\mathbf{H}_\alpha(\mathbf{P})$, $\mathbf{V}_\alpha(\mathbf{P})$ denote the 2D addition, 2D subtraction, and 2D multiplication of the two data planes **X**, **Y**, and horizontal and vertical shift of a data plane **P** respectively.

Mapping data-parallel algorithms

The key feature of data-parallel algorithms is that their parallelism comes from simultaneous operations across large sets of data, rather than from multiple thread of control.[1] A large portion of scientific computing algorithms fall into this category because of the enormous amounts of structured data that need to be processed. Various sources propose SIMD machines as the most suitable class of computers for dealing with these algorithms. These machines include image processing systems such as the MPP,[75] the Clip,[34] and the DAP 610,[78] as well as fine-grained parallel systems such as the Connection Machine.[79]

The SIMD optical architecture potentially offers a larger array size (larger number of processing elements) than existing counterparts. In addition, its unrestricted interconnections give it a greater flexibility in handling data-parallel

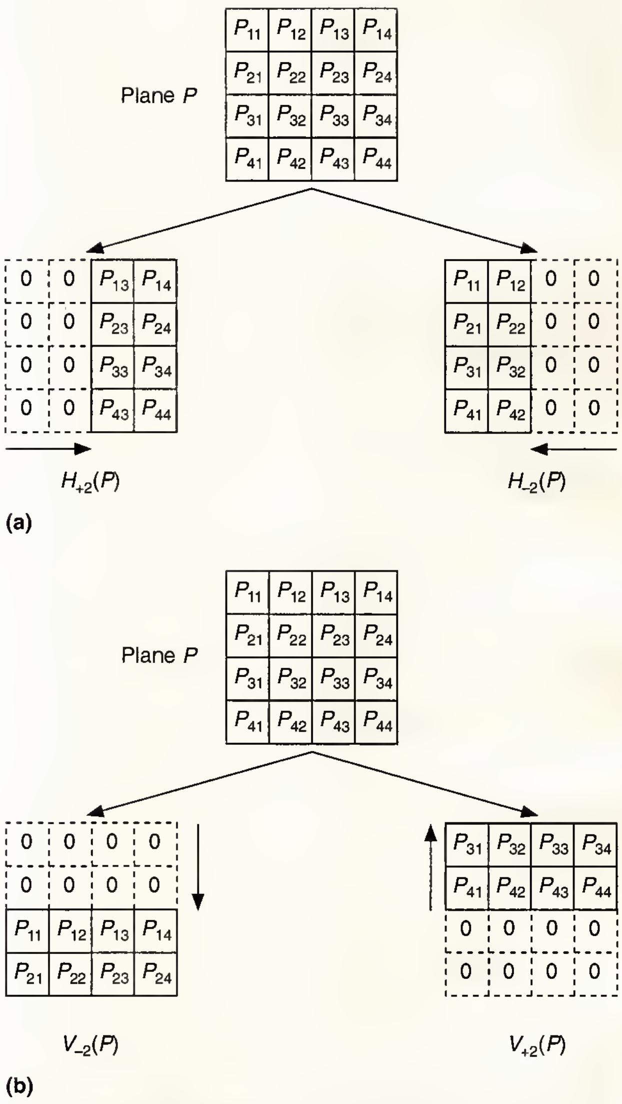

Plane P
P_{11} P_{12} P_{13} P_{14}
P_{21} P_{22} P_{23} P_{24}
P_{31} P_{32} P_{33} P_{34}
P_{41} P_{42} P_{43} P_{44}
$H_{+2}(P)$
$H_{-2}(P)$
(a)
$V_{-2}(P)$
$V_{+2}(P)$
(b)

Figure 12. Horizontal (a) and vertical (b) shift functions (shown for $\alpha = \pm 2$).

algorithms that require local as well as global communications. In the following, we show the mapping of several algorithms onto the architecture. We chose these algorithms to represent a broad range of complexity. They are also important key algorithms that occur as subproblems in larger programming tasks. Many more numerical algorithms have been mapped onto the optical architecture.[55]

Row/column accumulation. In calculating the sum of all the elements of a data plane columnwise (rowwise), we sum all the elements of a particular row; the final sum resides in the first entry of that row. Similarly, for column accumulation, we sum all the elements of a particular column, and the final sum occupies the first entry of that column. For a given data plane **S** of $n \times n$ elements, we proceed as follows. We split the initial plane **S** horizontally using the vertical shift operation (or vertically for rowwise accumulation using the horizontal shift operation) into two planes **X** and **Y**. Each plane contains half the data entries of **S**. Next we add these planes using the 2D addition substructure. We repeat this split-and-add process for $\log_2 n$ iterations, after which, the first row (first column) of **S** holds the accumulated sums of each column (row).

```
Procedure Row-Sum/Column-Sum(S,X,Y)
  begin
    for $k$ = 1 to $\log_2 n$ do
      $\alpha := n/2^k$;
      $\beta := \sum_{i=1}^{i=k} (n/2^k)$;
      $\mathbf{X} \leftarrow \mathbf{V}_{-\beta}(\mathbf{S})$      $(\mathbf{X} \leftarrow \mathbf{H}_{-\beta}(\mathbf{S})$ for Column-Sum);
      $\mathbf{V}_{+\beta}(\mathbf{X})$           $(\mathbf{H}_{+\beta}(\mathbf{X})$ for Column-Sum);
      $\mathbf{Y} \leftarrow \mathbf{V}_{+\alpha}(\mathbf{S})$      $(\mathbf{Y} \leftarrow \mathbf{H}_{+\alpha}(\mathbf{S})$ for Column-Sum);
      S ← 2D add(X,Y);
    endfor
  end Row-Sum/Column-Sum
```

We can combine the Row-Sum and Column-Sum substructures to compute the sum of all the elements of a data plane. To find the sum of the elements of a data plane, say **S**, we first apply the Row-Sum substructure to produce one column of accumulated sums. Next, we apply the Column-Sum substructure to accumulate the elements of that column. Figure 13 shows an example of computing the sum of the 16 elements stored in a 4×4 data plane **S**. We compute the sum after $2\log_2 16 = 8$ steps and store it in location $\mathbf{s}_{1,1}$. Similarly, we can compute the product of all the elements of a data plane (chain multiplication) using the 2D multiplication and vertical and horizontal shift substructures. The product of n^2 elements with q precision each can be found in $0(q^2 \log_2 n)$ time.

Matrix multiplication. Let **X**,**Y** be $n \times n$ matrices (assuming the same size for simplicity). Then their product $\mathbf{X} * \mathbf{Y} = \mathbf{Z}$ is an $n \times n$ matrix (the multiply asterisk denotes matrix multiply) whose elements are given by:

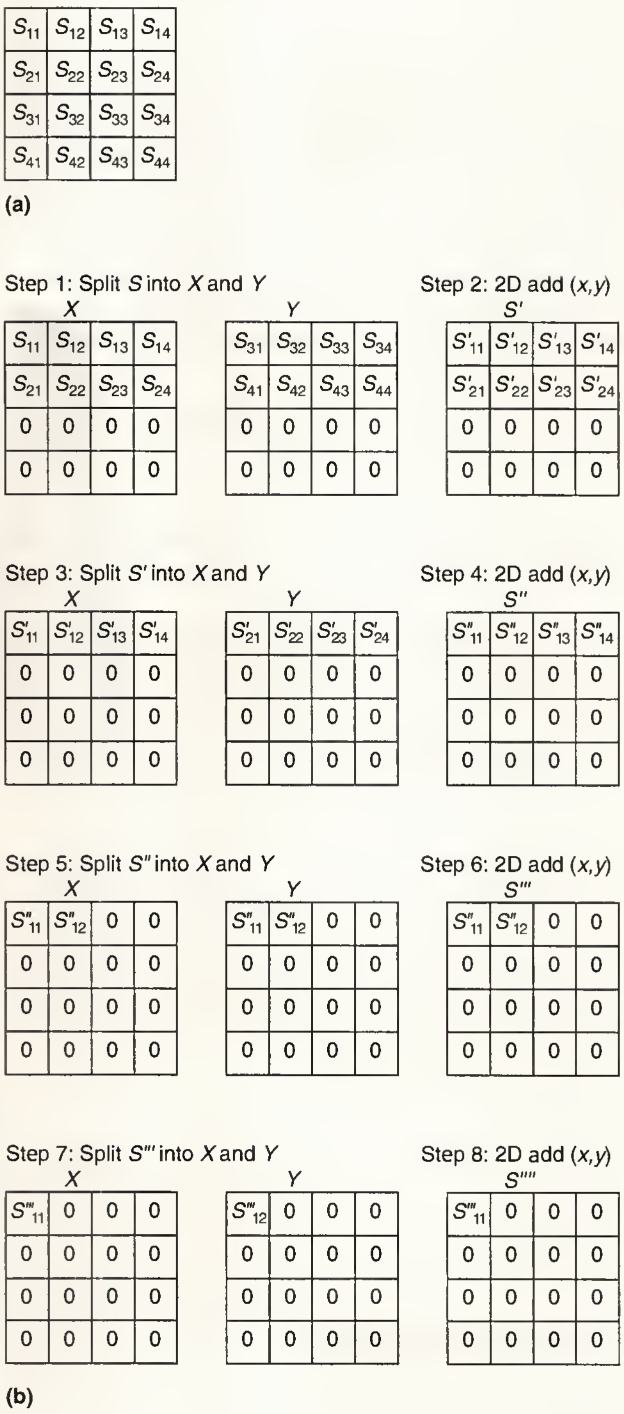

(a)
Step 1: Split S into X and Y
Step 2: 2D add (x,y)
Step 3: Split S' into X and Y
Step 4: 2D add (x,y)
Step 5: Split S'' into X and Y
Step 6: 2D add (x,y)
Step 7: Split S''' into X and Y
Step 8: 2D add (x,y)
X
Y
S'
S''
S'''
S''''
(b)

Figure 13. Sum of 16 integers using the 2D addition substructure on the optical architecture: set S of 16 integers (a) and summation algorithm (b). $S''''_{11} = \Sigma S_{i,j}$ for $i, j = 1, \ldots, 4$.

$$z_{ij} = \sum_{k=1}^{k=n} x_{ik} \times y_{kj}, \qquad i, j = 1, \ldots, n \qquad (4)$$

We assume matrix $\mathbf{X}$ is being transposed and stored as $\mathbf{X} = \mathbf{X}^t_1\mathbf{X}^t_2, \ldots, \mathbf{X}^t_n$, where $\mathbf{X}^t_i$ is an $n \times n$ matrix formed by replicating the ith column of the transposed matrix $\mathbf{X}^t$ n times. The basic approach for computing the jth row of the product matrix $\mathbf{Z}$ is to first generate the point-by-point multiplication of the elements of matrix $\mathbf{Y}$ by the elements of matrix $\mathbf{X}^t_j$, using the 2D multiplication substructure. Then we sum the columns of this matrix, using the Row-Sum substructure. As an example, consider matrices $\mathbf{X}$ and $\mathbf{Y}$ to be 3×3 of integers:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{bmatrix}$$

We assume matrix $\mathbf{X} = \mathbf{X}_1^t\,\mathbf{X}_2^t,\mathbf{X}_3^t$ where $\mathbf{X}_i^t$ is the ith row of matrix $\mathbf{X}$ transposed and replicated as follows:

$$\mathbf{X}_1^t = \begin{bmatrix} x_{11} & x_{11} & x_{11} \\ x_{12} & x_{12} & x_{12} \\ x_{13} & x_{13} & x_{13} \end{bmatrix} \quad \mathbf{X}_2^t = \begin{bmatrix} x_{21} & x_{21} & x_{21} \\ x_{22} & x_{22} & x_{22} \\ x_{23} & x_{23} & x_{23} \end{bmatrix} \quad \mathbf{X}_3^t = \begin{bmatrix} x_{31} & x_{31} & x_{31} \\ x_{32} & x_{32} & x_{32} \\ x_{33} & x_{33} & x_{33} \end{bmatrix}$$

The elementwise multiplication of $\mathbf{X}_1^t$ and $\mathbf{Y}$ using the 2D multiplication results in a matrix:

$$\mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix} = \begin{bmatrix} x_{11} \times y_{11} & x_{11} \times y_{12} & x_{11} \times y_{13} \\ x_{12} \times y_{21} & x_{12} \times y_{22} & x_{12} \times y_{23} \\ x_{13} \times y_{31} & x_{13} \times y_{32} & x_{13} \times y_{33} \end{bmatrix}$$

We accumulate the rows of matrix $\mathbf{T}$ using the Row-Sum substructure, to generate a matrix $\mathbf{Z}^1$ whose first row is the first row of the product matrix $\mathbf{Z}$:

$$\textbf{Row-Sum(T)} = \mathbf{Z}^1 = \begin{bmatrix} z^1_{11} & z^1_{12} & z^1_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where

$$z^1_{1j} = \sum_{k=1}^{k=3} t_{kj} = \sum_{k=1}^{k=3} x_{1k} \times y_{kj}.$$

In a similar manner, we use $\mathbf{X}_2^t$, and $\mathbf{X}_3^t$ to generate two matrices $\mathbf{Z}^2$ and $\mathbf{Z}^3$ respectively:

$$\mathbf{Z}^2 = \begin{bmatrix} z^2_{11} & z^2_{12} & z^2_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{Z}^3 = \begin{bmatrix} z^3_{11} & z^3_{12} & z^3_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where

$$z^2_{1j} = \sum_{k=1}^{k=3} x_{2k} \times y_{kj}$$

and

$$z^3_{1j} = \sum_{k=1}^{k=3} x_{3k} \times y_{kj} \text{ for } j = 1,2,3.$$

Note that the first row of $\mathbf{Z}^2$ and the first row of $\mathbf{Z}^3$ are the second and last rows of the product matrix $\mathbf{Z}$ respectively. We generate the product matrix $\mathbf{Z}$ by shifting $\mathbf{Z}^2$ by one row, and $\mathbf{Z}^3$ by two rows downward, and sequentially adding all the three matrices $\mathbf{Z}^1$, $\mathbf{Z}^2$, $\mathbf{Z}^3$ using the 2D addition substructure:

$$\mathbf{Z} = \begin{bmatrix} z^1_{11} & z^1_{12} & z^1_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ z^2_{11} & z^2_{12} & z^2_{13} \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ z^3_{11} & z^3_{12} & z^3_{13} \end{bmatrix}$$

The plus sign refers to the 2D addition substructure. The detailed algorithm is given as follows:

```
Procedure Matrix Multiply(Z,X,Y)
   begin
      for $k$ := 1 to $n$ do
           T ← 2D multiply( $X^k$,Y)
           $Z^k$ ← Row-Sum(T)
      endfor
      for $k$ := 1 to $n$ do
           $V_{(1-k)}(Z^k)$;
      endfor
      for $k$ := 1 to $n$ do
           Z ← 2D add(Z,$Z^k$);
      endfor
   end Matrix Multiply
```

The time complexity of the algorithm presented is $0[n(q\log_2 n + q^2)]$, where q is the operand length. Thus, the time complexity of this algorithm is $0(n\log_2 n)$, as opposed to $0(n^3)$ for the conventional triple-loop matrix multiplication.

Extrema finding. Extrema finding is the search for the maximum (minimum) value of a set of elements. The computations involved are global since the result is a function of all the data entries. These measures are very useful in many applications including signal/image processing, searching, and sorting.

Let the set of data be represented by a data plane $\mathbf{S}$ of n^2 elements. The algorithm for finding the maximum value of $\mathbf{S}$ proceeds by folding $\mathbf{S}$ repeatedly in half and selecting the largest value of overlapping elements from each half at each step. Initially we fold $\mathbf{S}$ in half by storing its first $n/2$ rows in the first $n/2$ rows of a matrix, say $\mathbf{S_u}$, and its last $n/2$ rows in the first rows of a second matrix $\mathbf{S_l}$. We subtract $\mathbf{S_l}$ from $\mathbf{S_u}$ using the 2D subtraction substructure, retain the maximum value of each pair of elements, and store it back into **S**. The new **S** contains half the data points of the original set.

We find the maximum value by repeating the folding and subtraction process for $2\log_2 n$. During the first $\log_2 n$ iterations, we fold the data plane **S** along the horizontal direction at each iteration. At the end of the first $\log_2 n$ iterations, each entry in the first row of **S** holds the maximum value of the corresponding column. Next, we fold **S** vertically and perform comparison for another $\log_2 n$ iterations after which the maximum value of the entire data plane **S** is located in the first entry $\mathbf{s_{11}}$.

```
Procedure Maximum(S,S_u,S_l)
  begin
    for $k$ = 1 to $\log_2 n$ do
      $\alpha := n/2^k$;
      $\beta := \sum_{i=1}^{i=k} (n/2^k)$;
      $\mathbf{S_u} \leftarrow \mathbf{V}_{-\beta}(\mathbf{S})$;
      $\mathbf{V}_{+\beta}\,\mathbf{S_u}$;
      $\mathbf{S_l} \leftarrow \mathbf{V}_{+\alpha}(\mathbf{S})$;
      $\mathbf{T} \leftarrow$ 2D sub($\mathbf{S_u}, \mathbf{S_l}$);
      if $T_{q-1} = 0$ then $\mathbf{S} \leftarrow \mathbf{S_u}$ else $\mathbf{S} \leftarrow \mathbf{S_l}$;
    endfor
    for $k$ = 1 to $\log_2 n$ do
      $\alpha := n/2^k$;
      $\beta := \sum_{i=1}^{i=k} (n/2^k)$;
      $\mathbf{S_u} \leftarrow \mathbf{H}_{-\beta}(\mathbf{S})$;
      $\mathbf{H}_{+\beta}(\mathbf{S_u})$;
      $\mathbf{S_l} \leftarrow \mathbf{H}_{+\alpha}(\mathbf{S})$;
      $\mathbf{T} \leftarrow$ 2D sub($\mathbf{S_u}, \mathbf{S_l}$);
      if $T_{q-1} = 0$ then $\mathbf{S} \leftarrow \mathbf{S_u}$ else $\mathbf{S} \leftarrow \mathbf{S_l}$;
  endfor          /*end k loop */
end Maximum
```

In this procedure, $\mathbf{T} = T_{q-1}T_{q-2}, \ldots, T_0$ is a temporary data plane used to hold the subtraction result. T_{q-1} is the most significant bit plane of **T**. Since we are using two's-complement subtraction, the most significant bit of the subtraction result indicates the relative magnitude of the operands. Each entry $T_{q-1}(ij)$ represents the sign bit of the subtraction operation $\mathbf{S_u}(ij) - \mathbf{S_l}(ij)$. Therefore, for $T_{q-1}(ij) = 0$, $\mathbf{S_u}(ij)$ is greater than or equal to $\mathbf{S_l}(ij)$; otherwise $\mathbf{S_l}(ij)$ is greater than $\mathbf{S_u}(ij)$. We achieve the selection of the largest value (noted by the simple conditional statement: if $T_{q-1} = 0$, then $\mathbf{S} \leftarrow \mathbf{S_u}$ else $\mathbf{S} \leftarrow \mathbf{S_l}$) by the following Boolean expression:

$$S_k = \text{P-And}(\text{P-Not}(T_{q-1}), S_{uk}) \vee \text{P-And}(T_{q-1}, S_l) \quad \text{for } k = 0, \ldots, q. \qquad (5)$$

where the sign $\vee$ is the logical Or and S_{uk}, S_{lk} refers to the kth-bit plane of arrays $\mathbf{S_u}$ and $\mathbf{S_l}$ respectively. We can express Equation 5 using only logical P-Not and P-And operators (using De Morgan's theorem $A \vee B = \overline{\overline{A} \wedge \overline{B}}$:

$$S_k = \text{P-Not (P-And (P-Not (P-And(P-not}(T_{q-1}), S_{uk})), \text{P-Not(P-And}(T_{q-1}, S_{lk})))) \qquad (6)$$

Several iterations through the system carry out Equation 6. The time complexity of the algorithm is $0(q\log_2 n)$. This algorithm is representative of many neighborhood algorithms such as those that find the minimum, the average, the median, the sum of a data plane, histogramming, counting, and so on. All can be implemented in $0(q\log_2 n)$ time. For example, we can apply the same algorithm to find the minimum of a set. In this case, we retain the minimum value at each iteration.

Projected performance

We estimated the theoretical performance of the optical architecture by evaluating several performance measures and compared them to the ones of existing SIMD array processors. The optical implementation of SSL follows the method briefly described earlier.[43,55]

We used the following key parameters in the analysis:

- T_p: Propagation time of a light beam through passive optical devices such as lenses, beam splitters, and holograms required for image replication and spatial shifting;
- T_{sw}: Response time of the optical switching devices such as the optical Nor-gate arrays, and the optical latches used in the processor array;
- T_{proc}: Response time of the optical processor array;
- T_{comb}: Response time of the input combiner;
- T_{rout}: Response time of the output router; and
- q: Precision of the operands (word length).

Optical cycle time. The optical cycle time, denoted by T_{array}, equals the time elapsed between inputting the data in the input planes and outputting the result at the output router. This time includes formatting the data at the input combiner, processing the formatted data in the processor array, and routing it to the appropriate destination. Therefore:

$$T_{array} = T_{comb} + T_{proc} + T_{rout} \qquad (7)$$

Please note that the definition of optical cycle time does not include memory access. Currently, we do not have a quantitative measure of response time of plane-addressable optical memories to include it in the analysis. The time needed to process data in the processor array (T_{proc}), using SSL hardware, is attributed to the time needed to:

1) deflect the formatted image to the active module (only one module can be active at a time);

2) replicate the formatted image, spatial shift the replicas, and combine the shifted copies;
3) activate the Nor-gate array for inverting the light intensity of the combined image;
4) propagate the inverted image though the mask;
5) replicate, shift, and combine images for the substitution phase; and
6) combine the outputs of all the activated SSL rules of the active module.

Using the above parameters, we derive T_{proc}:

$$T_{proc} = \overbrace{T_{sw}}^{(1)} + \overbrace{3T_p}^{(2)} + \overbrace{T_{sw}}^{(3)} + \overbrace{T_p + T_{sw}}^{(4)} + \overbrace{3T_p + T_{sw}}^{(5)} + \overbrace{T_p + T_{sw}}^{(6)} \quad (8)$$

The numbers over the braces indicate the times needed to accomplish each subtask as enumerated above. T_p can be in the range of 0.1 to 1 ns (light propagates at 1 ft/ns in free space). Presently, the status of active optical devices is much less mature than the passive components, and is the subject of intense research. The available optical switching devices have response times orders-of-magnitude higher than T_p (see Table 1). Thus, the dominant factor in Equation 8 becomes the switching time of the optical device used, T_{sw}. Consequently, we approximate T_{proc} as $T_{proc} = 5T_{sw}$.

The input combiner and the output router require optical switching devices along with some passive devices. We can also approximate the time spent in these units by the switching time of the active optical devices. We assume that $T_{comb} = T_{rout} = T_{sw}$. Then the total optical processing time is approximated by:

$$T_{array} = 7T_{sw} \quad (9)$$

Table 1 lists estimated values for the optical cycle time using different optical logic devices. The cycle time of the optical architecture is very much dependent upon the maturity of optical switching devices and their comparative performance with respect to electronic devices. Several approaches are being pursued for performing optical logic. One approach involves the adaptation of the SLM technology to optical logic. While current SLMs can be fabricated in large 2D arrays, their major drawback is the extremely slow response time (≥ 1 ms). However, considerable research is being performed to increase their spatial resolution (number of resolvable points on the array) and reduce their response time. The rationale is that large arrays of these devices will be fabricated to exploit the spatial parallelism of optics, and therefore compensate for their current slow response time.[24]

Another approach for realizing optical components capable of performing logic is to optimize the device from the beginning for digital operations. The recent emergence of the quantum-well self-electro-optic effect device—or SEED, and its derivatives (S-SEED, T-SEED, D-SEED)—is one such product.[18,19] We can use the SEED devices to realize both logic operations such as Nor, Or, And, and Nand as well as to store (S-R latches).[22] The family of SEED devices seems to be easy to use, capable of high-speed, low-energy operation, capable of being fabricated in 2D format, and cascadable.[36] Streibl et al.[80] reports that several hundred devices have been fabricated in 2D arrays on a chip and demonstrated with good uniformity. SEEDs show great potential for large parallel, digital optical computing systems (the optical architecture) because they are integrable low-energy devices (1-20 femtojoules per square centimeter), cascadable, and operate at high speed.

Table 1. Estimation of optical cycle time with respect to the physical characteristics of optical switching devices.*

Optical logic devices	Array size		Response time for entire array		Switching power		Cycle time (T_{array})	
	Available	Future	Available	Future	Available	Future	Available	Future
SLMs, LCLV[24,84]	500 × 500	500 × 500	20 ms	50 µs	µj/sq cm	NA**	140 ms	350 µs
Sight modulators[24,85]	128 × 128	1,024 × 1,024	1 µs	NA	NA	NA	7 µs	NA
FLC-SLMs[86,87]	400 × 400	10,000 × 10,000	155 µs	15 µs	0.1 pj/pixel	NA	1 ms	NA
SEED[19,22,23]	64 × 64 (within 10 years[13])	10,000 × 10,000	30 ns	1 ns	4 fj /sq µm	NA	210 ns	7 ns
Optical logic etalons[15,82,83]	100 × 100	10,000 × 10,000	150 ps	1 ps	100 pj	NA	1 ns	NA

* 1990 data
** Not available

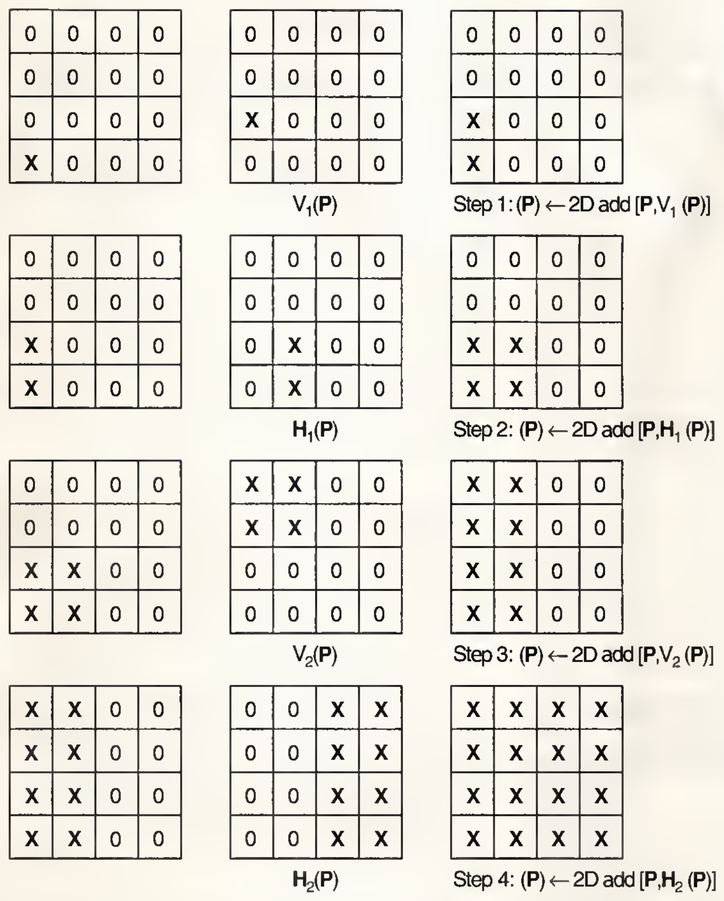

V$_1$(P)
Step 1: (P) ← 2D add [P,V$_1$ (P)]
H$_1$(P)
Step 2: (P) ← 2D add [P,H$_1$ (P)]
V$_2$(P)
Step 3: (P) ← 2D add [P,V$_2$ (P)]
H$_2$(P)
Step 4: (P) ← 2D add [P,H$_2$ (P)]

Figure 14. Broadcasting example on the optical architecture: The value *x* in the lower left-corner processing element is broadcast to all other PEs in the array in $2\log_2 4 = 4$ steps. The original plane **P** appears in the upper left array.

Optical resonators are another family under this approach intended for optical logic.[81,82] Two similar bistable devices, optical logic etalons (OLEs), and interference filters, both based on the Fabry-Perot resonator, are being actively pursued.[20,83] Although these devices can be fabricated in large 2D arrays ($10{,}000 \times 10{,}000$), and have a comparatively small response time, they are not cascadable at the present time.

OLEs are pulsed devices. In their operation, these devices require two wavelengths, one for the input signal and one for a bias signal (clock cycle). The two inputs, data and clock cycle, are separated in both time and wavelength. The modulated output signal has the same wavelength as the bias signal, and therefore the input and output signals have different wavelengths.[15] Hinton reports research efforts under way to implement a device composed of two OLE devices interconnected in such a way that the second OLE changes the wavelength of the output signal from the first device to the original input wavelength.[13]

Communication and I/O capabilities. Communication plays a crucial part in determining the performance of a parallel computer. Many communication metrics appear in the literature.[88,89] In the following analysis, we choose the most widely used.

The communication bandwidth is the maximum number of messages that can be simultaneously exchanged in one time step. Hence the bandwidth of the optical system is $0(n^2)$, since up to n^2 PEs can receive and send data at a time. Data transmits in the MPP and the Clip at one column at a time, therefore their bandwidth is $0(n)$. The DAP on the other hand transmits data in a row-parallel fashion, which amounts to the same bandwidth factor $0(n)$. The Connection Machine features a maximum sustained communication bandwidth of $0(n^2)$.

The diameter is the maximum number of communication cycles (or links) needed for any two PEs to communicate. For the optical case, this factor is 1, since we allow any number of shifts in either direction in one cycle time. The MPP and the DAP are mesh-connected and therefore have a diameter of $2(n - 1)$. The Clip has a hexagonal connectivity and therefore has a diameter of $n\sqrt{2}$. The Connection Machine is essentially wired in the pattern of a Boolean n cube, therefore its diameter is $0(\log_2 n)$.

Broadcasting sends the value in a certain PE to all the other PEs. The amount of communication cycles needed to achieve this is considered a measure of communication performance. This value is $0(n)$ for the MPP, DAP, and Clip, and $0(\log_2 n)$ for the Connection Machine. As far as the optical system is concerned, broadcasting a value in one PE to all other $n^2 - 1$ PEs takes $0(\log_2 n)$ steps. The example in Figure 14 shows the broadcasting of a value **x** residing in the lower left-corner PE to all other PEs in $2\log_2 4 = 4$ steps.

In current implementations of the MPP and the Clip, I/O processes in column-parallel fashion, while the row-parallel DAP loads data into the processor array one column or one row at a time. In contrast with the optical system, I/O activities process in a plane-parallel manner. This capability gives the optical system an I/O speedup of n, for an $n \times n$ input image, over the MPP, Clip, and the DAP. It could be a tremendous speed advantage, considering the large potential value of n. We note that the Connection Machine can also handle plane-parallel data, loading through a very expensive I/O system called the data vault.

Table 2 (on the next page) summarizes the various performance measures. Note that the processing time listed for the

Table 2. Performance comparison of the optical architecture with electronic array processors.

	Performance metrics				
Computing system	No. of PEs	Cycle time (ns)	Diameter	Bandwidth*	Broadcasting
Optical architecture	10,000 × 10,000 **	1	1	n^2	$0(\log_2 n)$
MPP[73]	128 × 128	100	$0(n)$	n	$0(n)$
DAP 610[78]	64 × 64	100	$0(n)$	n	$0(n)$
Clip[34]	96 × 96	2,500	$0(n)$	n	$0(n)$
Connection Machine[90]	65,536	100	$0(\log_2 n)$	n^2	$0(\log_2 n)$

*We assume the total number of PEs for the optical architecture to be the projected size of the SEED device.
**The bandwidth column refers to the communication bandwidth.

optical system is the estimated time when the implementation uses projected values for the SEED devices.[13]

Table 3 summarizes some performance measures for the parallel algorithms we have presented. These measures are theoretically evaluated by orders of magnitude only. From Table 3 we see that the theoretical efficiency of algorithms that are totally parallel and use every PE during the execution (such as 2D addition, 2D subtraction, and 2D multiplication) approaches 100 percent. (We assume the dimension of the problem matches the processor array size.) However, the theoretical efficiency is less than ideal for recursive algorithms because at each iteration only a fraction of the PEs are active. These algorithms include extrema finding, summation, and chain multiplication.

Table 3. Estimated performance of data-parallel algorithms on the proposed optical architecture.

Algorithm	Time complexity	Speedup*	Efficiency (speedup/ number of PEs)
2D addition ($n \times n$ arrays)	$O(q)$**	$\frac{qn^2}{q} = n^2$	1
2D multiplication ($n \times n$ arrays)	$O(q^2)$	$\frac{q^2n^2}{q^2} = n^2$	1
Sum of n^2 integers	$O(q\log_2 n)$	$\frac{q(n^2-1)}{q\log_2 n} \approx \frac{n^2}{\log_2 n}$	$\frac{1}{\log_2 n}$
Chain multiplication (n^2 number)	$O(q^2\log_2 n)$	$\frac{q^2(n^2-1)}{q^2\log_2 n} \approx \frac{n^2}{\log_2 n}$	$\frac{1}{\log_2 n}$
Matrix multiplication ($n \times n$ matrices	$O[n(q\log_2 n + q^2)]$	$\approx \frac{q^2n^3}{n(q\log_2 n+q^2)}$	$\approx \frac{q^2}{(q\log_2 n+q^2)}$
Maximum of n^2 integers	$O(q\log_2 n)$	$\frac{q(n^2-1)}{q\log_2 n}$	$\approx \frac{1}{\log_2 n}$

* The speedup is the ratio of the execution time on one 1-bit processing element to time taken on n^2 PEs.
** q is the precision (or operand length).

THE EVER-INCREASING DEMANDS for speed, throughput, and computing power, coupled with the limitations of electronic technology for massively parallel systems, brought about a major research impetus to explore optical technology for developing future massively parallel computers. Optics offers several advantages, including parallelism, high bandwidths, and noninterfering communications. Researchers seriously consider these advantages for breaking major bottlenecks imposed by conventional technology in storage, communications, and processing for high-performance computers. Justifications for the near-term inclusion of optics in parallel computing systems are already established for mass storage and interconnections. Optical disks begin to replace magnetic disks in some commercial systems. Moreover, optics is being used for processor-to-processor and processor-to-memory interconnections. Buses, optical backplanes, crossbar switches, reconfigurable interconnections (neural nets), and clock distribution networks are but a few applications of optical interconnections.

With the advances in optical storage and optical interconnections, optical information processing systems will have a major impact on overall system performance. If the data is already being stored and transmitted in optical form, optical computing processors might be the best alternative for processing the data rather than resorting to optical-to-electronic-to-optical conversions, which are major sources of performance degradation. Thus a more uniform technology for storage, communication, and processing of data in optical form will significantly impact the future of high-performance computing systems.

Digital optical information processing is the least developed at this time due to the immaturity of optical switching and logic devices. These devices are in their first generations. Considerable research and development efforts presently under way will lead to optical devices with lower switching energy, higher switching speeds, and higher resolutions. As these devices mature, optical computing systems will be highly competitive with existing electronic systems.

Here, we contribute to the ongoing efforts in building the foundations of new optical computing systems. We introduced a 3D optical computing architecture based on symbolic substitution logic. Although we focused on architectural and algorithmic issues plus some perfomance projections, we provide an extensive and up-to-date reference list that covers all aspects of the field. We showed that with a few symbolic substitution rules one can build a massively parallel optical computer. After introducing a hierarchical mapping technique for mapping parallel algorithms onto the optical computing model, we mapped several parallel algorithms onto the architecture. We chose these algorithms to represent a wide range of computational complexity.

We have assessed the theoretical performance of the proposed optical system. Although the system is not competitive at the present time with electronic array processors, it is quite attractive. It can potentially deliver a throughput at least 100 times higher than that of its electronic counterparts (owing to its multidimensional nature and high speed). Moreover, the communication flexibility and the parallel I/O of the optical system seems to be unmatchable with electronic array processors.

This preliminary performance analysis suggests that the proposed optical system is potentially a better alternative than current computing systems. The best applications are those that require the processing of large amounts of structured data such as remote sensing, signal/image processing, vision, weather modeling, and seismic data processing. We projected the performance under conservative assumptions and did not include power and budget breakdowns. A thorough power and cost analysis will require exact characteristics of the optical devices chosen for implementing different parts of the system. Such optical devices are just beginning to emerge. A more accurate assessment is needed once these optical devices are in wide use.

Acknowledgments

I thank the anonymous referees for their objectivity and constructive comments and gratefully acknowledge their help in the revision process. I also thank Kai Hwang for his valuable suggestions and technical review of the manuscript.

NSF Grant MIP-8909326 supported this research. I presented a preliminary version of this work at the 15th International Symposium on Computer Architecture in Honolulu on June 1, 1988.

References

1. W.D. Hillis and G.L. Steele, Jr., "Data Parallel Algorithms," *Comm. ACM,* Vol. 29, No. 12, Dec. 1986, pp. 1170-1183.
2. K. Hwang, "Advanced Parallel Processing with Supercomputer Architectures," *Proc. IEEE,* Oct. 1987, pp. 1348-1379.
3. G.S. Almasi and A. Gottlieb, *Highly Parallel Computing*, Addison Wesley, Reading, Mass., 1989.
4. A. Guha et al., "Optical Interconnections for Massively Parallel Architectures," *Applied Optics*, Vol. 29, Mar. 10, 1990, pp. 1077-1093.
5. A.A. Sawchuk and T.C. Stand, "Digital Optical Computing," *Proc. IEEE,* Vol. 72, No. 7, July 1984, pp. 758-779.
6. A. Huang, "Architectural Considerations Involved in the Design of an Optical Digital Computer," *Proc. IEEE,* Vol. 72, No. 7, July 1984, pp. 780-787.
7. P.B. Berra et al., "Optics and Supercomputing," *Proc. IEEE,* Vol. 77, Dec. 1989, pp. 1797-1815.
8. P.B. Berra et al., "The Impact of Optics on Data and Knowledge-Base Processing Systems," *IEEE Trans. Knowledge and Data Eng.,* Vol. 1, Mar. 1989, pp. 111-131.
9. W.T. Cathey et al., "Digital Computing with Optics," *Proc. IEEE,* Vol. 77, Oct. 1989, pp. 1558-1572.
10. J.W. Goodman et al., "Optical Interconnections for VLSI Systems," *Proc. IEEE,* Vol. 72, No. 7, July 1984, pp. 850-866.
11. A.A. Sawchuk et al., "Optical Crossbar Networks," *Computer*, Vol. 20, No. 6, June 1987, pp. 50-62.
12. L.D. Hutcheson and A. Husein, "Optical Interconnections Replace Hardware," *IEEE Spectrum*, 1987, pp. 30-35.
13. H.S. Hinton, "Architectural Considerations for Photonic Switching Networks," *IEEE J. Selective Areas in Comm.*, Vol. 6, Aug. 1988, pp. 1209-1226.

14. Special issue, "Optical Interconnections," *Applied Optics*, Vol. 29, Mar. 1990.
15. J.L. Jewell et al., "3-pj, 82-MHz Optical Logic Gates in a Room Temperature GaAs-AlGaAs Multiple Quantum-Well Etalon," *Applied Physics Letters*, 1985, pp. 918-925.
16. H.M. Gibbs et al., "Optical Bistable Devices: The Basic Components of an All-Optical System," *Optical Eng.*, Vol. 19, 1980, pp. 463-468.
17. A.R. Tanguay, "Material Requirements for Optical Processing and Computing Devices," *Optical Eng.*, Vol. 24, No. 1, Jan./Feb. 1985, pp. 2-17.
18. D.A.B. Miller et al., "Large Room-Temperature Optical Nonlinearity in GaAs/$Ga_{1-x}Al_xAs$ Multiple Quantum Well Structures," *Applied Physics Letters*, Vol. 44, No. 10, 1982.
19. D.A.B. Miller et al., "The Quantum Well Self-Electro-optic Effect Device: Opto-electronic Bistability and Oscillation, and Self-Linearized Modulation," *IEEE J. Quantum Electronics*, Vol. QE-21, Sept. 1985, pp. 1462-1476.
20. S.D. Smith, "Room-Temperature, Visible-Wavelength Optical Bistability in ZnSe Interference Filters," *Optics Comm.*, Vol. 51, Oct. 1984, pp. 357-362.
21. G. Livescu et al., "Spatial Light Modulator and Optical Dynamic Memory Using a 6×6 Array of Self-Electro-optic Effect Devices," *Optics Letters*, Vol. 13, Apr. 1988, pp. 297-299.
22. A.L. Lentine et al., "Symmetric Self-Electro-optic Effect Device: Optical Set-Reset Latch, Differential Logic Gate, and Differential Modulator/Detector," *IEEE J. Quantum Electronics*, Vol. 25, Aug. 1989, pp. 1928-1936.
23. A.L. Lentine et al., "Optical Logic Using Electrically Connected Quantum Well PIN Diode Modulators and Detectors," *Applied Optics*, Vol. 29, No. 14, May 10, 1990, pp. 2153-2163.
24. J.A. Neff et al., "Two-Dimensional Spatial Light Modulators: A Tutorial," *Proc. IEEE*, Vol. 78, May 1990, pp. 836-855.
25. B.K. Jenkins et al., "Sequential Optical Logic Implementation," *Applied Optics*, Vol. 23, No. 19, Oct. 1984, pp. 3473-3474.
26. K.S. Huang et al., "Optical Cellular Logic Architectures Based on Binary Image Algebra," *Proc. Workshop on Computer Architecture Pattern Analysis and Machine Intelligence*, Oct. 1987, pp. 19-26.
27. J. Tanida and Y. Ichioka, "Optical Logic Array Processor Using Shadowgrams," *J. Optical Soc. Am.*, Vol. 73, No. 6, June 1983, pp. 800-809.
28. T.J. Drabik and S.L. Lee, "Shift-Connected SIMD Array Architectures for Digital Optical Computing Systems, with Algorithms for Numerical Transforms and Partial Differential Equations," *Applied Optics*, Vol. 25, No. 22, Nov. 1986., pp. 4053-4064.
29. F. Kiamilev et al., "Programmable Opto-electronic Multiprocessors and Their Comparison with Symbolic Substitution for Digital Optical Computing," *Optical Eng.*, Vol. 28, No. 4, 1989, pp. 396-409.
30. J. Taboury et al., "Optical Cellular Logic Architecture 1: Principles," *Applied Optics*, Vol. 27, No. 9, May 1, 1988, pp. 1643-1650.
31. D.P. Casasent and E.C. Botha, "Multifunctional Optical Processor Based on Symbolic Substitution," *Optical Eng.*, Vol. 28, No. 4, Apr. 1989, pp. 425-433.
32. K.H. Brenner, "Programmable Optical Processor Based on Symbolic Substitution," *Applied Optics*, Vol. 27, No. 9, May 1, 1988, pp. 1687-1691.
33. B.S. Werrett et al., "Construction and Tolerancing of an Optical-CLIP," *Proc. SPIE*, Vol. 1215, 1990, pp. 264-273.
34. M.J. Duff, *CLIP 4: Special Computer Architecture for Pattern Recognition*, Fu and Ichikawa, eds., CRC Press, 1982.
35. M.J. Murdocca et al., "Optical Design of Programmable Logic Arrays," *Applied Optics*, Vol. 27, May 1988, pp.1651-1660.
36. M. Prise et al., "Module for Optical Logic Circuits Using Symmetric Self-Electro-optic Effect Devices," *Applied Optics*, May 10, 1990, pp. 2164-2170.
37. M. E. Prise, "Optical Computing Using Self-Electro-optic Effect Devices," *Proc. SPIE*, Vol. 1214, Jan. 1990.
38. H.J. Caulfield and G. Gheen, eds., "Selected Papers on Optical Computing," *SPIE Milestone Series*, Vol. 1142, 1989.
39. A.D. Mc Aulay, *Optical Computing Architectures: The Application of Optical Concepts to Next-Generation Computers*, John Wiley & Sons, New York, 1990.
40. D.G. Feitelson, *Optical Computing: A Survey for Computer Scientists*, MIT Press, Cambridge, Mass., 1988.
41. A. Huang, "Parallel Algorithms for Optical Digital Computers," *Proc. IEEE 10th Int'l Optical Computing Conf.*, 1983, pp. 13-17.
42. H.S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. Computers*, Vol. C-20, No. 2, 1971, pp. 153-161.
43. K.H. Brenner et al., "Digital Optical Computing with Symbolic Substitution," *Applied Optics*, Vol. 25, No. 18, Sept. 15, 1986, pp. 3054-3060.
44. Y. Li et al., "An And Operation-Based Optical Symbolic Substitution," *Optics Comm.*, Vol. 63, No. 6, Sept. 15, 1987, pp. 375-379.
45. F.T.S. Yu and S. Jutamulia, "Implementation of Symbolic Substitution Logic Using Optical Associative Memories," *Applied Optics*, Vol. 26, No. 12, June 15, 1987, pp. 2293-2294.
46. E. Botha et al., "Optical Symbolic Substitution Using Multichannel Correlators," *Applied Optics*, Vol. 27, No. 5, Mar. 1, 1988, pp. 817-818.
47. J.N. Mait and K.H. Brenner, "Optical Symbolic Substitution: System Design Using Phase-Only Holograms," *Applied Optics*, Vol. 27, No. 9, May 1, 1988, pp. 1692-1700.
48. K.H. Brenner, "New Implementation of Symbolic Substitution Logic," *Applied Optics*, Vol. 25, No. 18, Sept. 15, 1986, pp. 3061-3064.
49. A. Louri, "Efficient Implementation Method for Symbolic Substitution Logic Based on Shadow-Casting," *Applied Optics*, Vol. 28, No. 15, Aug. 15, 1989, pp. 3264-3267.
50. A. Louri, "Throughput Enhancement for Optical Symbolic Substitution Computing Systems," *Applied Optics*, Vol. 29, No. 20, July 10, 1990, pp. 2979-2981.
51. H.J. Jeon et al., "Digital Optical Processor Based on Symbolic Substitution Using Holographic Matched Filtering," *Applied Optics*, Vol. 29, May 10, 1990, pp. 2113-2125.

52. K.H. Brenner et al., "Symbolic Substitution Implemented by Spatial Filtering Logic," *Optical Eng.*, Vol. 28, No. 14, 1989, pp. 390-396.
53. A. Louri, "Parallel Implementation of Optical Symbolic Substitution Logic Using Shadow-Casting and Polarization," *Applied Optics,* Feb. 1991, pp. 540-548.
54. A. Louri, "Parallel Implementation of Optical Symbolic Substitution Logic Using Shadow-Casting and Wavelength Multiplexing," submitted to *Optics Letters*.
55. A. Louri and K. Hwang, "A Bit-Plane Architecture for Optical Computing with 2D Symbolic Substitution Algorithms," *Proc. 15th Int'l. Symp. Computer Architecture*, May 1988, pp. 18-27.
56. A.W. Lohmann et al., "Optical Perfect Shuffle," *Applied Optics*, Vol. 25, No. 10, May, 15 1986, pp. 1530-1531.
57. A.A. Sawchuk and B.K. Jenkins, "Dynamic Optical Interconnections for Optical Processors," *Proc. Soc. Photo-Optical Instrumentation Engineers,* Jan. 1986, Vol. 625.
58. C.W. Stirk et al., "Folded Perfect Shuffle Optical Processor," *Applied Optics*, Vol. 27, No. 2, Jan. 1988, pp. 202-203.
59. H.H. Brenner and A. Huang, "Optical Implementations of the Perfect Shuffle Interconnections," *Applied Optics,* Vol. 27, No. 1, Jan. 1988, pp. 135-137.
60. Y. Sheng and H. Arsenault, "Light Effective Perfect Shuffle Using Fresnel Mirrors," *Tech Digest, Topical Meeting on Optical Computing*, Vol. 9, Feb. 1989, pp. 154-157.
61. A.W. Lohmann, "What Classical Optics Can Do for the Digital Optical Computer," *Applied Optics*, Vol. 25, No. 10, May 15, 1986, pp. 1543-1549.
62. P.B. Berra et al. "Optical Database/Knowledge Base Machines," *Applied Optics*, Vol. 29, Jan. 10, 1990, pp. 195-205.
63. C. Warde and A. Fisher, "Spatial Light Modulators: Applications and Functional Capabilities," *Optical Signal Processing*, J. Horner, ed., Academic Press, New York, 1987, pp. 478-524.
64. A.D. Fisher and J.N. Lee, "Current Status of Two-Dimensional Spatial Light Modulators," *Proc. SPIE Optical and Hybrid Computing*, Vol. 634,1986, pp. 352-372.
65. S.D. Smith, "Optical Bistability, Photonic, Logic and Optical Computation," *Applied Optics*, Vol. 25, May 15, 1986, pp. 1550-1564.
66. L. Solymar and D. J. Cooke, *Volume Holography and Volume Gratings*, Academic Press, New York, 1981.
67. G.C. Valley and M.B. Klein, "Optimal Properties of Photorefractive Materials for Optical Data Processing," *Optical Eng.*, Vol. 22, Dec. 1983.
68. D. Psaltis et al., "Adaptive Optical Networks Using Photorefractive Crystals," *Applied Optics*, Vol. 27, No. 9, May 1988, pp. 1752-1759.
69. D. Psaltis et al., "Optical Neural Nets Implemented with Volume Holograms," *Tech. Digest, OSA Topical Meeting on Optical Computing*, 1987, pp. TuA3.1-TuA3.4.
70. S. Redfield and L. Hesselink, "Data Storage in Photorefractives Revisited," *SPIE Proc. Optical Computing 88*, J.W. Goodman et al., eds., Vol. 963, 1988, pp. 35-45.
71. R.P. Freese, "Optical Disks Become Erasable," *IEEE Spectrum*, Vol. 25, Feb. 1988, pp. 41-45.
72. T. Murakami et al., "Magneto-optic Erasable Disk Memory with Two Optical Heads," *Applied Optics*, Vol. 25, Nov. 1986, pp. 3986-3989.
73. D. Psaltis et al., "Parallel Readout of Optical Disks," *Tech. Digest, Topical Meeting on Optical Computing,* Vol. 9, Feb. 1989, pp. 58-62.
74. P.D. Henshaw and A. Todtenkopf, "Artificial Intelligence Applications of Fast Optical Memory Access," *Proc. SPIE, Optical and Hybrid Computing*, Vol. 634, 1986, pp. 422-438.
75. S. Hunter et al., "Potentials of Two-Photon-Based 3D Optical Memories for High-Performance Computing," *Applied Optics*, Vol. 29, May 10, 1990, pp. 2058-2066.
76. K. Hwang, "Computer Architecture for Image Processing," *Computer*, Vol. 16, Jan. 1983, pp. 10-12.
77. K.E. Batcher, "Design of a Massively Parallel Processor," *IEEE Trans. Computers*, Vol. C-29, Sept. 1980, pp. 836-884.
78. C.G. Winckless, "Massively Parallel Computer for Signal and Image Processing," *Proc. IEEE Int'l Symp. Circuits and Systems,* May 1989, pp. 1396-1399.
79. W.D. Hillis, *The Connection Machine*, MIT Press, Cambridge, Mass., 1985.
80. N. Streibl et al., "Digital Optics," *Proc. IEEE*, Vol. 77, Dec. 1989, pp. 1954-1970.
81. H. Gibbs and N. Peyghambarian, "Nonlinear Etalons and Optical Computing," *Proc. SPIE, Optical and Hybrid Computing*, Vol. 634, Mar. 1986, pp. 142-148.
82. J.L. Jewell et al., "GaAs-AlAs Monolithic Microresonator Arrays," *Applied Physics Letters,* Vol. 51, July 1987, pp. 94-96.
83. J.L. Jewell et al., "Use of a Single Nonlinear Fabry-Perot Etalon as an Optical Logic Gate," *Applied Physics Letters,* Vol. 44, Jan. 1984, pp. 172-174.
84. M.S. Welkowski et al., "Status of Charge-Coupled-Device-Addressed Liquid Crystal Light Valve," *Optical Eng.*, Vol. 26, 1987, pp. 414-417.
85. B. Hill, "The Current Status of Two-Dimensional Spatial Light Modulators," *Optical Computing: Digital and Symbolic*, R. Arrathoon, ed., Marcel Dekker, New York, 1989, pp. 1-40.
86. G. Moddel et al., "High-Speed Binary Optically Addressed Spatial Light Modulator," *App. Phys. Lett.*, Aug. 1989.
87. K.M. Johnson and G. Moddel, "Motivations for Using Ferroelectric Liquid Crystal Spatial Light Modulators in Neurocomputing," *Applied Optics*, Vol. 28, Nov. 1989, pp. 4888-4899.
88. T.Y. Feng, "A Survey of Interconnection Networks," *Computer*, Vol. 14, Dec. 1981, pp. 12-27.
89. S.P. Levitan, "Measuring Communications Structures in Parallel Architectures and Algorithms," *The Characteristics of Parallel Algorithms*, Chap. 4, L.H. Jamieson et al., eds., MIT Press, 1987, pp. 101-139.
90. "Connection Machine Model CM-2 Technical Summary," Tech. Report Series HA87-4, Thinking Machine Corporation, Cambridge, Mass., 1987.

Ahmed Louri is an assistant professor in the Department of Electrical and Computer Engineering at the University of Arizona. Previously, he worked with the Computer Research Institute at the University of Southern California in Los Angeles, conducting research into parallel processing, multiprocessor system design, and optical computing. His technical interests include high-speed parallel processing, optical computing, and optical interconnections.

Louri earned the Engineer Degree diploma in electrical engineering from the University of Science and Technology, Algeria, and MS and PhD degrees in computer engineering from USC. He is a member of the IEEE, Association of Computing Machinery, Optical Society of America, and the Society of Photo-Optical Instrumentation Engineers.

Address questions regarding this article to the author at the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721; or e-mail at louri@pairs.ece.arizona.edu.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 159 Medium 160 High 161

Micro News

continued from p. 10

Micro bits

Texas Instruments and joint-venture partner **Kobe Steel** plan to fabricate 8-inch silicon wafers containing advanced logic chips at a Japanese manufacturing site. Production of CMOS, VLSI, and ASIC devices for distribution and sale to TI customers should begin in 1992.

An IEEE **Posix** seminar scheduled for April 29-30 in San Diego and June 10-11 in Washington, D.C., will explain the standard and introduce methods for creating profiles that meet the needs of an organization. Call (908) 562-3805 for data.

MIT will offer an eight-week course in **technical Japanese** for computer scientists and electrical engineers beginning June 10.

A Business Communications Company study suggests that advanced fine-line, **multilayer PCBs** will be the preferred option for electronic packaging into the next century. The PCB worldwide market could total $31.9 billion by the year 2000, up from $13.5 billion in 1990.

The recently opened **Garage museum** at the Technology Center of Silicon Valley in San Jose, Calif., lets visitors participate in an interactive media lab and view exhibits of microelectronics, robotics and materials, space exploration, biotechnology, and high-tech bicycling.

LSI Logic, Milpitas, Calif., announces publication of *Fast Forward.* The bimonthly newsletter's inaugural issue of on video compression will be followed by issues featuring ASIC test alternatives and design-for-test developments.

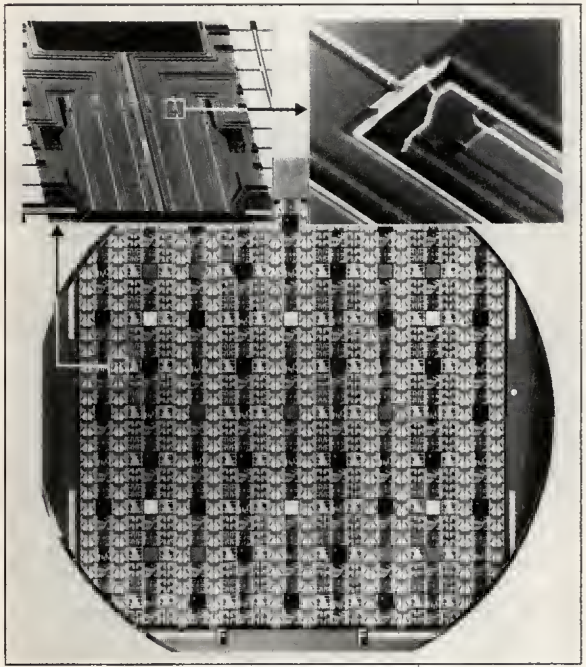
Tiny lasers

IBM's Zurich Research Laboratory reports fabrication and testing of 5,000 to 20,000 1/32- and 1/64-inch, AlGaAs crystal lasers on a wafer 2 inches in diameter.

The company uses its full-wafer technology to etch 1/5,000-inch-deep trenches into the wafer to produce convex and concave laser mirrors. Next, the mirrors are coated with a semireflective material to improve reliability. As electrical current passes through the lasers, the lasers emit invisible, infrared light in 830- and 850-nanometer wavelengths that are usable in reading and writing optical data. The mirrors enhance and direct the path of the 30- and 40-milliwatt light generated from 1/10,000-inch apertures.

IBM grows the lasers from crystals of aluminum gallium arsenide using standard epitaxy methods. It then adds dopants to control electronic properties.

Photodiodes and other test components on the wafer monitor laser performance. According to IBM, process characterization, diagnostics, and screening for working lasers can all be accomplished on the wafer.



IBM's laser wafer (main photo). The two enlarged views show an array of six individual lasers and accompanying photo diodes (left) and a single laser and photo diode (right).

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 198 Medium 199 High 200

Software Report

continued from p. 11

The physical organization would be hierarchical, with 5-cm-square chips (neurons) organized 64 to a board. Boards would be connected via a 3 × 3 grid at the subsystem level, stacked on planes into a system, and systems connected together somehow. He called this a "recursively modular architecture" and felt it would be a CMOS MPP supercomputer.

He claimed that a 1-million-neuron system with 2 tera updates (multiplication and addition) per second (Tcups) is definitely within range. For example, Hitachi has already built via wafer scale integration (WSI) a board containing eight wafers with 100 neurons per wafer. Okabe feels the technology of the larger neuron system can be built using a 20-million-transistor chip in 1995, with 6-inch wafers and 60 chips per wafer. The neuron circuit would be completely digital, with a learning algorithm using back propagation and with an 8-bit data I/O bus, and 8-bit weights. (It was pointed out during several sessions that for significant numerical computation it will be necessary to have at least 32-bit capability.) This assumes about 120,000 transistors per neuron. A 1-million-neuron system would consist of 1,000 subsystems each composed of 1,000 fully connected neurons. He even showed a slide of an entire system on one 50 × 70-cm board, composed of two rows of 50 WSI cards. Each 20 × 20-cm card would contain a 6-inch wafer with 10,000 neurons.

Massively parallel systems. I found myself confused by what the speakers actually meant by massively parallel systems. Some interpretations include "only neural computing," "genetic computing," and "dataflow computing." The interest is clearly not in the type of massive parallelism that has resulted in the Connection Machine. There was no discussion of supercomputing or biological computing. I asked if "soft" computing related to fuzzy logic. One Japanese scientist admitted that he felt the latter was a "quick-and-dirty" approach and that NIPT should look at much more fundamental ideas. None of the others disputed that statement.

Software details appeared sketchy except that work needs to be done on parallel languages and that the computational model is probably going to be a combination of object-oriented and concurrent programming.

By the year 2000 we should expect 20 million transistors on an 8-inch wafer. A billion-cell system is not impossible.

Oyanagi of Toshiba stated that by the year 2000 we should expect 20 million transistors on an 8-inch wafer. There would be 100,000 transistors per cell, 200 cells per chip, 30 chips per wafer, 1,000 wafers per stack, and 16 stacks per system. He estimated that four stacks could be built on a 100 × 100-cm board, and that a 10^8-cell system would have to dissipate about 160 kW of power. Thus heat dissipation would be a significant problem. Nevertheless he felt that building a *billion*-cell system would not be impossible. He then went on to describe a three-dimensional implementation using printed circuits and VLSI (not WSI) that Matsushita is building.

It will implement about 1 million cells in one cubic meter. Power dissipation is around 320 kW, cooled by heat pipes. He concluded with a table (see Table 1) describing two target systems, one for 1995 and another for 2000.

Once again, the hardware issues seem very much clearer than the software issues, and Oyanagi acknowledged this afterwards. Also, there was no suggestion of the software and design issues related to reliability and fault tolerance of such huge systems.

Future technologies. An informal evening session included discussions concerning Hitachi, Fujitsu, and Toshiba neural network research projects. Hitachi representatives described a 2.3-Gcups neuro system built with eight 5-inch wafers, 144 neurons per wafer (30,000 transistors per neuron), using a 0.8-μm CMOS gate array. The system measures 30 × 21 × 23 cm and dissipates about 50 watts.

The interesting thing about this system in addition to its speed (about four times faster than any previously built) is that the weights and connections are dynamically changeable and that the weight values can be full 16 bits. The speed comes from a clever use of two separate buses, one each in the input and output directions. Learning is implemented via back propagation. Several applications have already been programmed including signature verification and stock prediction. Hitachi formally announced this device three or four days before the workshop.

Fujitsu representatives offered an overview of their neurocomputing research activities. They have built, or are working on, three different systems including a PC board. The most interesting is Sandy, a collection of 256 Texas Instrument floating-point digital signal processors on a ring network. Each DSP presently functions as one neuron, but Fujitsu claims that the software can allow each to function as four (or more) neurons. Because of the DSP, honest 32-bit floating-point operations are possible.

Currently an eight-processor prototype is running; a 256-processor version will be capable of 6 Gcups for back propagation. Several applications were cited, including mobile robot control, stock forecasting, and convertible bond rating. More practical is a process failure-prediction system to be used during steel continuous casting. It determines the "breakout time," the time at which the cooling process fails. Fujitsu is also experimenting with combining a neural net with a fuzzy-reasoning interface.

Toshiba described a 512-processor system organized as an 8 × 8 × 8 set of crossbar switches representing the faces of a 3-cube. In other words, any processor can communicate with any other in at most three hops. There were no other details given, and the current status of the project was not made clear. Work is also continuing on a Japanese word processor that uses a neural network to select kanji from kana input.

International cooperation. A late-morning combined session titled "Toward International Cooperation" offered formal presentations by the Japanese and informal presentations by attendees from the US, the European Community, and the German national research center for computer science (GMD). The ICOT program did not emphasize international cooperation to the extent that this new program appears to do.

Deputy Director Taiso Nishikawa expressed the MITI (Ministry of International Trade and Industry) commitment to international cooperation. He stated directly that no concrete proposal exists as yet, and that time and discussion as well as ideas are needed. A hand-drawn overhead transparency of the cooperation plan was greeted by good-natured howls because of its complexity as well as warm support for the speaker's willingness to present it.

Nishikawa proposed that a central office should control a pool of funds supplied by Japan, the US, and the European Community. MITI and counterpart government organizations in the other participating countries should be involved. Laboratories should be located in all participating countries, with an exchange of scientists. The central pool should also fund international consortia. The whole organization should have complete symmetry with respect to national borders. It should operate under the US/Japan agreement on cooperation in science and technology and a counterpart EC/Japan agreement. Intellectual property rights should be allocated according to the degree of participation and in accordance with the agreements.

Eugene Wong presented the US government view. He explained that the US government officials at this meeting came as observers, not participants, and were attracted by the prominence of international cooperation on the agenda. He recognized many theoretical advantages of cooperation, such as economy in the use of research and development funds that could be used for other means of promoting economic growth in a time of worldwide capital shortage. But he pointed out that excellence is driven by competition, that Japan has learned better than the US that cooperation and competition can coexist, and praised MITI for fostering both successfully.

H.W. Muehlenbein from GMD presented the European perspective and commented that funding in the European Community's ESPRIT program is much larger than NIPT's is likely to be. (The major European Community research funding agency is the European Strategic Program for Information Technology, which controls a billion-dollar budget spent mostly on short-term research in electronic computing.)

Muehlenbein remarked that European Community members are experienced in international cooperation, and it works well for them because ESPRIT has plenty of funds. Without that funding, there would be more conflict than cooperation. He emphasized a point that was taken up by other university researchers, that without the promise of new funds they will not be interested in participation. He advised the governments involved in this initiative to concentrate on that aspect of the organization.

T. Hagemann of GMD-Tokyo defined the issues as: What to do, what to do with the results (intellectual property rights), and how to do it (organization and funding). The property rights issue should be clarified at the very beginning to avoid headaches later on. He suggested that distributing property rights to the contributing researchers was the best mechanism.

[David Kahaner is on assignment with the US Office of Naval Research, Far East. His comments are his own; they do not express any official policy.]

Table 1. MPP targets.

Parameters	1995 (silicon)	2000
Design rule	0.3-0.5 µm	0.13-0.2 µm
Integration	WSI or VLSI	WSI
Transistors/cell	100,000 to 1,000,000	1,000,000 to 1,000,000,000
Purpose	Testbed, software development	Integrated system
Environment	Optical network	Optical network

Reader Interest Survey
Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 195 Medium 196 High 197

Read *IEEE Micro*'s June issue for...

...articles from the Computer Society TCMM's Hot Chips II Symposium featuring:

- Intel's 100-MOP LIW iWarp microprocessor for multicomputers
- ITT Intermetall's Data Wave single-chip video multiprocessor
- IBM's RISC System/6000: Architecture and performance
- Intergraph's Clipper C100-C400 chipset architecture
- Metaflow Technology's Metaflow architecture

FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

Northern California and Pacific Northwest: John D. Vance & Associates, Inc., 4030 Moorpark Ave., Suite 116, San Jose, CA 95117; (408) 741-0354.

Southern California and Mountain States: Richard C. Faust Co., 24050 Madison St., Suite 101, Torrance, CA 90505; (213) 373-9604.

Midwest: The Kingwill Company, 4433 W. Touhy Ave., Suite 540, Lincolnwood, IL 60091; (708) 675-5755.

East Coast: Atlantic Representative Group, 349 Maple Place, Keyport, NJ 07735; (908) 739-1444.

New England: Arpin Associates, P.O. Box 6444, Holliston, MA 01746; (508) 429-8907.

Europe: Heinz J. Görgens, Parkstrasse 8a, D-4054 Nettetal 1 - Hinsbeck (F.R.G.); phone: (0 21 53) 8 99 88; fax: (0 21 53) 8 99 89.

Southwest/Southeast: Heidi Rex, Office, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; (714) 821-8380.

For production information, conference, and classified advertising, contact Heidi Rex or Marian Tibayan.

IEEE MICRO, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; phone (714) 821-8380; fax (714) 821-4010.

	RS #	**Page #**
Alsys	20	37
Burr-Brown Corp.	12, 84	36, 39
Computer Safety Products	11	36
Converter Concepts	16	37
Dilog	24	38
Esker	30	38
General Datacomm	28	38
Integrated Device Technology	80	39
Intelligent Interfaces	25	38
JMI Software Consultants	19	37
KEA Systems	31	38
Language Systems	22	37
Martech	26	38
Microdata Soft	17	37
Mitsubishi Electronics America	83	39
Motorola	18, 85	37, 39
The NIT Group	29	38
Nanotek	27	38
Pacific Rim Systems	15	37
Philips Components	82, 86	39
Promised Land Technologies	21	37
Tektronix	10	36
Telebyte Technologies	14	36
Texas Instruments	81	39
Tradewinds Peripherals	13	36
Xycom	23	37

Micro View

continued from p. 88

compatible chip but is making a multichip set that implements the 386 architecture. Nexgen's implementation strategy resembles that used for LSI Logic's Lightning Sparc processor: The chip set uses techniques including out-of-order execution, branch prediction, and register renaming. Nexgen is rumored to be as much as two years behind its original schedule. The last chip of the set went into fabrication late last year, and Nexgen hasn't made any public announcements—other than replacing its president—since then.

Nexgen is under considerable pressure, since its key goal is to provide higher performance than Intel's single-chip implementation. Nexgen has stated that it expects to offer at least twice the performance of Intel's 486. With Intel moving the 486 to 50 MHz next year, however, it will become increasingly difficult for Nexgen to provide a significant performance boost.

If Nexgen is able to get its chip set to work as planned, it will possess a valuable piece of technology. In another year or two, it will be feasible to put the current multichip design onto one chip. This development could make Nexgen a contender for the 586-class microprocessor market.

Other clones

Industry observers believe several other companies are working on 386-compatible processors. The only one that has publicly stated its intentions is Integrated Information Technology, which sells Intel-compatible coprocessors and VGA/8514 display controllers. IIT said that it plans to develop a 486-compatible chip, but it has not given a time frame. It seems unlikely that it will ship such a product in 1991.

Rumors have circulated for years that Chips and Technologies is developing a 386-compatible processor, but the company will neither confirm nor deny the existence of such a project. C & T has shown customers under nondisclosure a chip that combines a fast, 8086-compatible implementation with a display controller and PC system logic. The firm also may be close to announcing a math coprocessor.

A Japanese semiconductor company, V.M. Technology Corp., is shipping (in Japan) a microprocessor that implements the real-mode 386 instruction set but does not include the memory management unit or protected-mode logic. The company may be developing a full 386-compatible chip and appears to have the technical ability to do so, but it will not comment on these reports. Fujitsu provides foundry services to V.M. and is a major investor.

Meridian Semiconductor, a small design house in Orange County, Calif., has completed a logic design for a superscalar processor that is pin- and software-compatible with the 386. The company has built a TTL emulator to verify the design, and currently seeks a semiconductor partner to help turn the design into a chip.

Intel's best defense

While Intel's dominance will fade, it is not in danger of losing its leading position.

All of the clone makers face possible legal challenges from Intel. Intel has patents on several aspects of the 386 architecture that are difficult to avoid while maintaining compatibility. AMD's patent license gives it a distinct advantage in this regard. Microcode copyrights are relatively easy to avoid by developing clean-room microcode.

Even if AMD and others ship 386-compatible products in volume this year, Intel will remain in a strong position. Intel has a broader product line—including the 386DX, 386SX, 386SL, 486, and the rumored 486SX—than any of its competitors. Many of Intel's 386 customers also make 486-based systems, so availability of 386 chips from AMD does not free them from dependence on Intel. Because of this, system vendors are wary of damaging their relationship with Intel and are likely to move slowly in adopting non-Intel processors. Their concern is that Intel would give preferential treatment, including early shipments of new chip versions, to customers that remained loyal. Some call this paranoia, but remember, just because you're paranoid doesn't mean they're not out to get you.

Intel's best defense is the continuing proliferation of the product line. By moving the 486 up to 50 MHz, it will be difficult for others to beat its performance. Because Intel has years of experience in manufacturing the 386, its chips are smaller than those of any likely competitor. Intel recently revealed its new 0.8-µm 486 implementation, demonstrating its ability to continue to reduce die size and increase clock rates. Intel could compete aggressively on price, should the need arise, while maintaining higher margins than its competitors.

Intel's strongest defense against 386 cloners is the processor code-named P23, which some sources expect to debut as the 486SX. This processor is essentially a 486 without a floating-point unit. If Intel prices this chip only slightly above the price of the 386, it is likely to cause most new system designs to shift away from that processor, just as AMD moves into volume production.

The industry will benefit from the multivendor status the 386 will achieve this year, and Intel has a strong defense ready to keep itself from being badly hurt.

Reader Interest Survey

Please indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 201 Medium 202 High 203

Micro View

The end of the 386 monopoly

Michael Slater

Microprocessor Report

(707) 823-4004

mslater@cup.portal.com

With AMD's shipment of 386-compatible microprocessors, the most lucrative monopoly in the history of the semiconductor business is ending. Intel's position remains strong, however, and the continued growth of the market will make it possible for Intel to prosper at the same time that it loses some market share to AMD and others.

The 386 is over five years old, and it is remarkable that Intel's monopoly has lasted this long. The difficulties of cloning the 386 are considerable, and AMD is the only company that appears likely to ship 386-compatible chips this year. AMD is in a unique position because it has an undisputed license to Intel's patents and, in AMD's view, it also has a license to Intel's microcode. This licensing position allowed AMD to produce a close copy of Intel's logic design. Intel disputes AMD's right to use Intel's microcode, and this issue is scheduled to go to trial this summer (see Micro View, Dec. 1990). If AMD loses, it will have to substitute clean-room microcode for the Intel microcode—a contingency for which AMD is probably prepared.

AMD has been sampling its Am386DX to about 20 customers since last November and claims to have found no compatibility problems. AMD expects to begin shipping production quantities about the time this column appears. Early production runs will go entirely to key customers, and parts will not show up in distribution until the second half of the year. AMD has fabricated first silicon on an SX version as well, and this part will lag behind the DX by about three months.

AMD will not pursue an aggressive pricing strategy as it has with the 287 math coprocessor. In the case of the coprocessor, two non-Intel competitors were already shipping chips, and the market is such that lowering the price could significantly increase demand. With the 386, however, Intel has been unable to meet demand, so AMD can sell a substantial number of chips even if it prices them identically to Intel's.

AMD offers its 386DX in a low-power version that not only draws one-third less power than Intel's chip at its maximum clock rate, but also can operate at arbitrarily slow clock rates to reduce power consumption. AMD also plans to offer a low-power version of its 386SX, which, unlike Intel's 386SL, can drop into existing designs. The fastest growing segment of the personal computer business is notebook computers based on the SX. Since AMD's chip will give these computers a longer battery life than Intel's 386SX, AMD should have great success selling its 386SX chips to notebook computer makers.

The appearance of AMD's 386 chips in systems is likely to be largely unheralded. System manufacturers have no incentive whatsoever to publicize the fact that they are using AMD chips. This publicity could only serve to alienate Intel—which remains an important supplier to these companies—and cause concern among its customers. From the customers' point-of-view, it shouldn't matter whose 386 chip is in the system. No doubt system vendors will perform extensive compatibility testing before using AMD's chips. AMD is offering a 40-MHz version, while Intel's top speed is 33 MHz. Therefore, the most conspicuous AMD customers will be those desiring the 40-MHz systems.

Nexgen

Nexgen Microsystems is another company that is frequently mentioned when the subject of 386 clones arises. Nexgen is not developing a pin-

continued on p. 87

0272-1732/91/0400-0088$01.00 © 1991 IEEE

1951-1991

40 YEARS OF SERVICE

IEEE COMPUTER SOCIETY

A member society of the Institute of Electrical and Electronics Engineers, Inc.

THE FOLLOWING INFORMATION IS AVAILABLE:

Contact the Publications Office;
to facilitate handling, please request by number.

- Membership application, student #203, others #202
- Periodicals subscription form for individuals #200
- Periodicals subscription form for organizations #199
- Publications catalog #201
- Compmail electronic mail brochure #194
- Technical committee list/application #197
- Chapters lists, start-up procedures #193
- Student scholarship information #192
- Volunteer leaders/staff directory #196
- IEEE senior member grade application #204
  (requires ten years practice and significant performance in five of those ten)

To check membership status or report a change of address, call the IEEE toll-free number, 1-800-678-4333. Direct all other Computer Society related questions to the Publications Office.

PURPOSE

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 100,000 members worldwide, and provides a wide range of services to members and nonmembers.

MEMBERSHIP

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

PUBLICATIONS AND ACTIVITIES

Computer. An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field, plus news, conferences, calendar, interviews, and product reviews.

Periodicals. The society publishes six magazines and five research transactions. Refer to membership application or request information as noted above.

Conference Proceedings, Tutorial Texts, Standard Documents. The Computer Society Press publishes more than 100 titles every year.

Standards Working Groups. Over 100 of these groups produce IEEE standards used throughout the industrial world.

Technical Committees. More than 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

Conferences/Education. The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

Chapters. Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

OMBUDSMAN

Members experiencing problems — magazine delivery, membership status, or unresolved complaints — may write to the ombudsman at the Publications Office.

EXECUTIVE COMMITTEE

President:	Duncan H. Lawrie* University of Illinois Dept. of Computer Science 1304 W. Springfield Urbana, IL 61801 (217) 333-3373
President-Elect:	8ruce D. Shriver*
Past President:	Helen M. Wood*
VP, Standards:	Paul L. 8orrill (1st VP)*
VP, Press Activities:	Barry W. Johnson (2nd VP)*
VP, Conferences and Tutorials:	Laurel V. Kaleda†
VP, Education:	Raymond E. Miller†
VP, Membership Activities	Ronald D. Williams†
VP, Publications:	Ronald G. Hoelzeman†
VP, Technical Activities:	Mario R. 8arbacci*
Secretary:	James H. Aylor*
Treasurer:	Joseph 8oykin†
Division V Director:	Edward A. Parrish, Jr.†
Division VIII Director:	Helen M. Wood*
Executive Director:	T. Michael Elliott†

*voting member of the Board of Governors
†nonvoting member of the Board of Governors

BOARD OF GOVERNORS

Term Expiring 1991:
P. 8ruce 8erra, Michael Evangelist,
Ted Lewis, Raymond E. Miller, Earl E. Swartzlander, Jr.,
Joseph E. Urban, Thomas W. Williams

Term Expiring 1992:
James H. Aylor, Alicja I. Ellis, Tadao Ichikawa,
C.V. Ramamoorthy, Sallie V. Sheppard,
Harold Stone, Akihiko Yamada

Term Expiring 1993:
Fiorenza Albert-Howard, Jon T. 8utler, Michael C. Mulder,
Yale N. Patt, Anneliese von Mayrhauser,
Benjamin W. Wah, Ronald Waxman

Next Board Meeting
May 17, 1991, 8:30 a.m.
Stouffer Austin Hotel, Austin, Texas

SENIOR STAFF

Executive Director:	T. Michael Elliott
Publisher:	H. True Seaborn
Director, Conferences and Tutorials:	Anne Marie Kelly
Director, Finance and Administration:	Tod S. Heisler
Director, 8oard and Administrative Services:	Violet S. Doan

COMPUTER SOCIETY OFFICES

Headquarters Office
1730 Massachusetts Ave. NW
Washington, DC 20036-1903
Phone (202) 371-0101
Fax: (202)728-9614

Publications Office
10662 Los Vaqueros Cir.
PO Box 3014
Los Alamitos, CA 90720-1264
Membership and General Information:
(714) 821-8380
Publication Orders: (800) 272-6657
Fax: (714) 821-4010

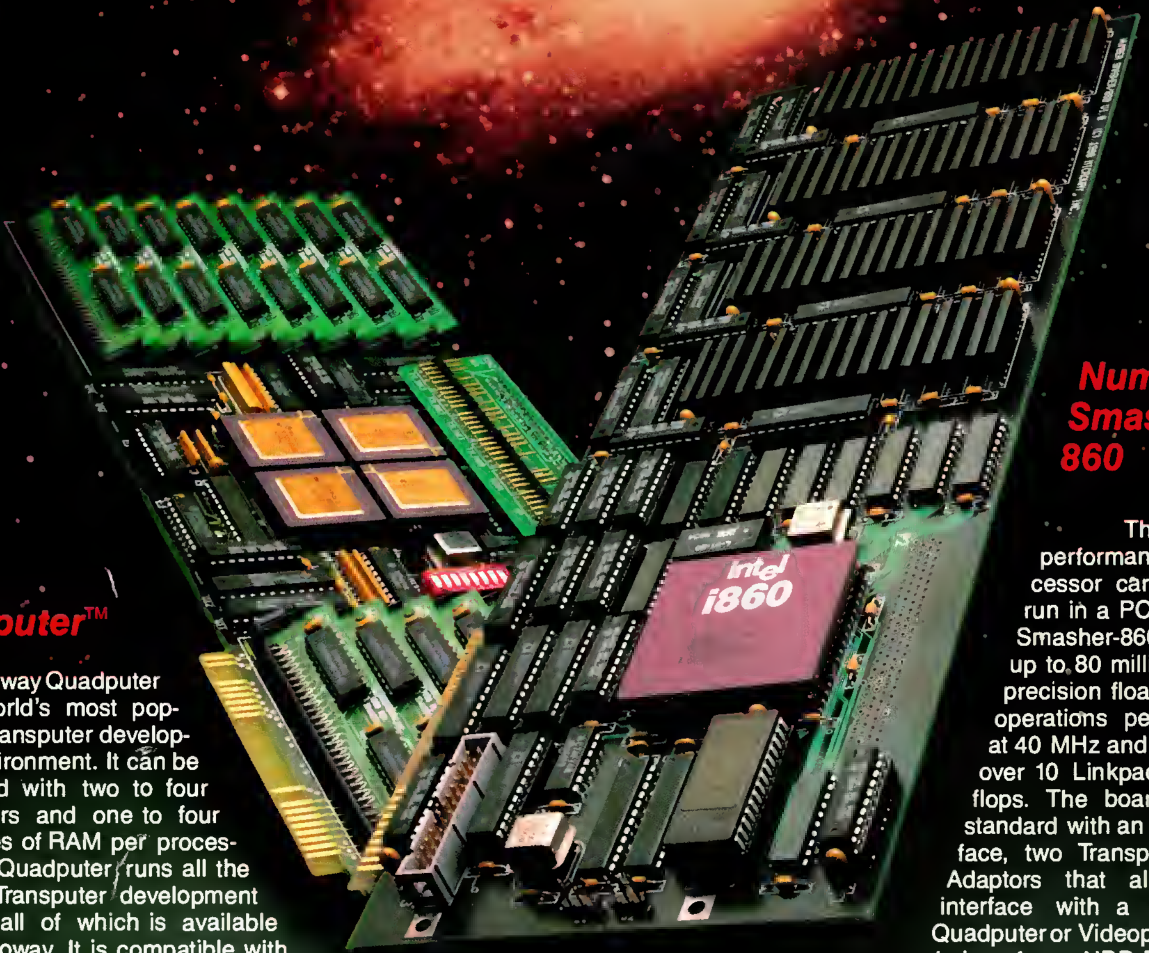
European Office
13, Ave. de L'Aquilon
8-1200 8russels, Belgium
Phone: 32 (2) 770-21-98
Fax: 32 (2) 770-85-05

Asian Office
Ooshima 8uilding
2-19-1 Minami-Aoyama, Minato-ku
Tokyo 107, Japan
Phone: 81 (3) 3408-3118
Fax: 81 (3) 3408-3553

IEEE OFFICERS

President:	Eric E. Sumner	VP, Educational Activities:	Richard S. Nichols
President Elect:	Merrill W. Buckley, Jr.	VP, Professional Activities:	Michael J. Whitelaw
Past President:	Carleton A. 8ayless	VP, Publication Activities:	J. Thomas Cain
Secretary:	Hugh Rudnick	VP, Regional Activities:	Robert T. H. Alden
Treasurer:	Theodore W. Hissey, Jr	VP, Technical Activities:	Fernando Aldana

Discover Parallel Processing



Quadputer™

The Microway Quadputer is the world's most popular PC Transputer development environment. It can be purchased with two to four Transputers and one to four megabytes of RAM per processor. The Quadputer runs all the popular Transputer development software, all of which is available from Microway. It is compatible with our Monoputer™ which provides 1to16 megabytes of RAM and a single T800, our Videoputer™ which comes in VGA and higher resolution versions and is powered by a memory mapped pair (T800 and 34010), and our Linkputer™ whose cross bar switching network can dynamically link up to 32 Transputers. Finally, all Microway Transputer products can be used with our Number Smasher-860 to provide out-of-this-world numeric performance!

For more information, please call 508-746-7341.

Number Smasher® 860

The highest performance coprocessor card to ever run in a PC, Number Smasher-860 delivers up to 80 million single precision floating point operations per second at 40 MHz and produces over 10 Linkpack megaflops. The board comes standard with an ISA interface, two Transputer Link Adaptors that allow it to interface with a Microway Quadputer or Videoputer, your choice of our NDP Fortran, C or Pascal for the 80860, plus 8 megabytes of high speed memory.

NDP Fortran-860, C-860 and C++860

Microway NDP 860 Compilers make it easy to recompile your favorite mainframe, 80386 or PC applicaton for the 80860. The resulting code runs on our XTEND-860™ environment under DOS, UNIX or XENIX.

Microway

The World Leader in PC Numerics

Corporate Headquarters, Research Park, Box 79, Kingston, MA 02364
TEL 508-746-7341 • FAX 508-746-4678
U.K. - 32 High St., Kingston-Upon-Thames, 081-541-5466 • Italy 02-74.90.749
Holland 40 836455 • Germany 069-75-2023 • Japan 81 3 222 0544